Abstraktion von Benutzungsschnittstellen auf Basis der Extensible Markup Language (XML)

MyXaml zur Realisierung von Skinning am Beispiel eines ESG-Client-Prototypen

Diplomarbeit

zur Erlangung des akademischen Grades

Diplom-Informatikerin (FH)

an der

Fachhochschule für Technik und Wirtschaft Berlin

Fachbereich Wirtschaftswissenschaften II

Studiengang Internationale Medieninformatik

1. Betreuer: Prof. Thomas Bremer

2. Betreuerin: Prof. Dr. Debora Weber-Wulff

Eingereicht von Merrit Kröger

Matrikelnummer: 76900502130

Berlin, 04. Juli 2005

Alle Rechte vorbehalten

Inhaltsverzeichnis

1. Einleitung....1
2. Hintergrund....5
2.1. DVB-H-Projekt....5
2.2. Vorstellung des Prototypen....9
2.3. Technische Vorgaben....11
3. Grundlagen....13
3.1. Begriffserklärung....13
3.2. Pocket PC....14
3.3. Benutzungsschnittstellen....16
3.4. Auszeichnungssprachen....19
3.4.1. XML....21
3.4.2. XAML....22
3.4.3. MyXaml....25
4. Analyse....30
4.1. Aufgabenstellung....30
4.2. Definition der Zielgruppe....31
4.3. Auswahl eines Lösungsansatzes....31
4.4. Bestehender ESG-Client....36
4.4.1. Systemarchitektur....36
4.4.2. Analyse ausgewählter Komponenten....37
4.4.2.1. ESG-Client....38
4.4.2.2. IESGPlugIn....41
4.4.2.3. TVZapPlugIn....42
4.4.2.4. ImageButton....46
4.4.3. Auswertung der Analyse der bestehenden Anwendung....47
4.5. Anwendungsfälle....49
4.6. Resultierende Anforderungen an die Anwendung....52
4.6.1. Allgemeine Anforderungen....52
4.6.2. Anforderungen hinsichtlich der „Skinningfähigkeit“....53
4.6.3. Anforderungen hinsichtlich der „Skinning-Durchführung“....54
4.7. Zieldefinition....54
5. Re-Design....56
5.1. Systemarchitektur....56
5.1.1. Trennung von GUI und Applikationslogik....56
5.1.2. Anpassung des ImageButtons....58
5.1.3. Anpassung der Plug-In-Schnittstelle....60
5.1.4. Package-Diagramm der modifizierten Anwendung....61
5.1.5. Entwurf des Skinning-Features....62
5.2. Stylesheets und Skins....63
5.3. Funktionalität....64

Alle Rechte vorbehalten

6. Implementierung....67
6.1. Vorgehensweise....67
6.1.1. Phase 1 – Umsetzung der Skinningfähigkeit....67
6.1.2. Phase 2 – Durchführbarkeit des Skinnings....73
6.1.3. Zusammenfassung....75
6.2. Problembeschreibungen....77
6.3. Testergebnisse....81
7. Ergebnisse....86
8. Zusammenfassung....91
Anhang....94
A - Stilzuweisungen (zu 5.2.)....94
B - Klassendiagramme des resultierenden ESG-Client (zu 6.1.3.)....96
C - Abkürzungsverzeichnis....99
D - Glossar....100
E - Abbildungsverzeichnis....103
F - Tabellenverzeichnis....105
G - Literaturverzeichnis....106
H - CD-ROM....111
I - Danksagung....112
J - Eigenständigkeitserklärung....113

Alle Rechte vorbehalten

1. Einleitung

In der Softwareentwicklung wurde sich bereits seit vielen Jahren ausgiebig mit der Problematik der Trennung der einzelnen Bestandteile von Softwareprodukten auseinandergesetzt. Es wird noch heute darauf hin gearbeitet, verschiedene Tätigkeiten logisch voneinander abzugrenzen um u.a. die Möglichkeit einer verteilten Entwicklung durch Designer und Programmierer zu schaffen. Anwendung findet diese Idee bereits im *MVC-Design-Konzept*, bei dessen Gebrauch eine Applikation in ein zugrunde liegendes Datenmodell (Model), eine Ansichtsebene (View) und eine Kontrollebene (Controller) zerlegt wird.

Ein konkreter Bedarf an einer Trennung von Darstellung und zugehöriger Applikationslogik besteht bei der Implementierung eines *Electronic Service Guide-Client* (ESG-Client) für das digitale mobile Fernsehen (DVB-H, *Digital Video Broadcasting-Handheld*), welches in naher Zukunft per Broadcast an mobile Empfangsgeräte, z.B. *Personal Digital Assistants* (PDA's), übertragen wird. Der ESG-Client stellt neben einer Art Programmzeitschrift auch den einzigen Zugang zu den Diensten dieses digitalen mobilen Fernsehens dar. Ein Prototyp einer solchen Anwendung wurde im Wintersemester 2004/05 an der Fachhochschule für Technik und Wirtschaft (FHTW) in Berlin im Rahmen einer Projektarbeit von Studenten der Internationalen Medieninformatik in Zusammenarbeit mit der T-Systems International GmbH Media & Broadcast (TSI) erstellt. Dieses Projekt wird im gegenwärtig laufenden Sommersemester 2005 weitergeführt und soll voraussichtlich im September 2005 auf der Internationalen Funkausstellung in Berlin präsentiert werden.

Das Design des prototypischen ESG-Client ist ein erster Entwurf und kann sich bis zu dessen Veröffentlichung noch mehrere Male ändern. Zudem ist es von der TSI geplant, die Anwendung an andere Firmen, z.B. Netzbetreiber oder Anbieter digitaler Inhalte, zu verkaufen. Aus diesem Grund muss das Design der grafischen Benutzungsschnittstelle auf das *Corporate Design* (CD), d.h. Logos, Farben, Schriften etc., der jeweiligen Firma abgestimmt werden können, ohne die unterliegende Funktionsweise des ESG-Client zu beeinflussen und den Programmieraufwand dabei möglichst gering zu halten. Denkbar wäre ebenso eine benutzerdefinierte Gestaltung der Darstellungsebene durch den Endkunden selbst, um den Spaß im Umgang mit der Anwendung zu gewährleisten. Dieser Forderung nach individueller Anpassbarkeit des bestehenden prototypischen ESG-Client wird in der vorliegenden Diplomarbeit nachgegangen. Dazu soll u.a. die vollständig in der Programmiersprache C#

Alle Rechte vorbehalten

implementierte Anwendung in zwei Bereiche, die Definition der Ansicht und die zugehörige Programmlogik, unterteilt werden.

Ein zunehmend verwendeter Lösungsansatz zur Realisierung einer solchen Trennung der Darstellungsebene von der unterliegenden Applikationslogik, ist die Beschreibung der *grafischen Benutzungsschnittstelle* (GUI, Graphical User Interface) mittels Textauszeichnung (engl. *Markup*) auf Basis der *Extensible Markup Language* (XML). Das bedeutet, dass die Definition der sichtbaren Bestandteile der Anwendung anstelle von Programmcode in Form von textuellen Elementen in einem separaten Dokument vorliegt. Dieses XML-Dokument wird von einem Programm, einem so genannten *Parser*, gelesen, die darin enthaltenen Informationen verarbeitet und die benötigten Objekte erzeugt. Microsoft implementiert dieses Konzept vollständig im kommenden Betriebssystem Windows „Longhorn“ durch Verwendung der neu entwickelten Auszeichnungssprache XAML (*Extensible Application Markup Language*).

Ein XML-basierter Lösungsansatz ermöglicht eine Verwendung von *Skins* (engl. für Stilsammlungen, die u.a. Farben und Bilder definieren), wie sie bereits bekannt ist aus Anwendungen wie dem Windows Media Player, dem MP3-Player WinAmp oder den Anwendungen des Mozilla Projektes, wie z.B. dem Internetbrowser Firefox. Diese Softwareprodukte bieten die Möglichkeit der benutzerdefinierten Gestaltung der GUI mit Hilfe von vorgefertigten oder selbst erstellten Dokumenten, welche die Darstellungsweise der sichtbaren Elemente definieren. Durch Austausch dieser Dokumente lässt sich einfach und schnell das Aussehen der GUI verändern. Die besondere Problematik, die sich im Falle einer individuellen Anpassbarkeit des ESG-Client-Prototypen stellt, ist, dass alle bekannten Technologien, die eine Trennung der Darstellung von der Programmlogik ermöglichen, wie z.B. die Mozilla-eigene Sprache XUL (*XML User Interface Language*) oder Microsoft's XAML, nicht für PDA's verfügbar sind. Demzufolge können diese nicht zur Modifikation des ESG-Client verwendet werden, da dieser auf der Plattform *Pocket PC* – eine Kategorie von PDA's – lauffähig sein muss.

MyXaml ist ein neuartiger Lösungsansatz, der bereits die Verwendung der Markup-Sprache XAML ermöglicht und zudem für die Entwicklung von Software für die Plattform Pocket PC benutzt werden kann. Bei dieser Technologie handelt es sich um eine Implementierung, mit deren Hilfe die Bestandteile der grafischen Benutzungsschnittstelle ebenfalls in Form von Textauszeichnung beschrieben sowie diese Elemente vom zugehörigen Parser zur Laufzeit der Anwendung erzeugt werden können. Sämtliche Applikationslogik

Alle Rechte vorbehalten

kann in Form einer beliebigen, zur Entwicklung für die Pocket PC-Plattform geeigneten Programmiersprache vorliegen.

Das Ziel dieser Diplomarbeit besteht darin, die Definition der grafischen Benutzungsschnittstelle des prototypischen ESG-Client in die XML-basierte Sprache MyXaml zu überführen und unter Verwendung von Skins die Darstellung der sichtbaren Elemente der Anwendung benutzerdefiniert austauschbar zu machen, so dass ein *Branding*, d.h. eine Anpassung in der Gestaltung der Elemente an das Corporate Design, für verschiedene Firmen möglich ist. Lauffähig soll die modifizierte Anwendung in erster Linie auf der Pocket PC-Plattform sein. Weiterhin soll ein einfaches Werkzeug entwickelt werden, mit dessen Hilfe die erreichte individuelle Anpassungsfähigkeit des ESG-Client getestet und demonstriert werden kann.

Die einzelnen Kapitel der vorliegenden Diplomarbeit werden im Folgenden kurz vorgestellt:

Kapitel 2 – Hintergrund

Zunächst werden die bestehende Anwendung und das Projekt vorgestellt, in dessen Rahmen der zu modifizierende ESG-Client-Prototyp entstand. Weiterhin wird auf technische Vorgaben eingegangen, die aus dem Projekt resultieren und im Rahmen dieser Arbeit ebenfalls Gültigkeit besitzen.

Kapitel 3 – Grundlagen

In diesem Kapitel soll näher auf die in dieser Arbeit angewendeten Technologien eingegangen werden. Zunächst wird die verwendete Terminologie erklärt. Im Anschluss daran wird das grundlegende Wissen zu den Themenbereichen Pocket PC, Benutzungsschnittstellen und Auszeichnungssprachen, insbesondere XML, XAML und MyXaml, vermittelt.

Kapitel 4 – Analyse

Gegenstand dieses Kapitels sind eine allgemeine Aufgabenbeschreibung und die Festlegung der Zielgruppe für das neue System, sowie die Suche und Auswahl eines zur Modifikation verwendbaren Lösungsansatzes. Unter dem Gesichtspunkt der Verwendung der gewählten Lösung soll daraufhin die bestehende Anwendung untersucht sowie Anwendungsfälle zur geplanten Durchführung der benutzerdefinierten Anpassung beschrieben werden. Anhand der Analyse werden zum einen Anforderungen abgeleitet, die an das veränderte System zu stellen sind, sowie zum anderen das konkrete Ziel definiert, welches im Rahmen der vorliegenden Arbeit erreicht werden soll.

Alle Rechte vorbehalten

Kapitel 5 – Re-Design

Gegenstand des fünften Kapitels ist die Anfertigung eines konkreten Entwurfes des neugestalteten ESG-Client. Dieser umfasst eine Definition der zukünftigen Systemarchitektur, die Vorgaben zum Aufbau der zu verwendenden Skins sowie eine Beschreibung der Funktionalität der modifizierten Anwendung.

Kapitel 6 – Implementierung

Auf Basis des Entwurfes erfolgt die Implementierung des neuen Systems. Die Vorgehensweise bei deren Durchführung wird in diesem Kapitel beschrieben sowie während dessen aufgetretene Probleme benannt und erläutert. Anhand eines Tests der realisierten Funktionalität sollen Fehler in der Implementierung aufgedeckt und ausgemerzt werden.

Kapitel 7 – Ergebnisse

Die im Rahmen dieser Arbeit erzielten Resultate werden in Kapitel 7 zusammengetragen, mit der Zielstellung verglichen und bewertet. Ebenso werden alternative Vorschläge zur Umsetzung der implementierten Funktionalität unterbreitet.

Kapitel 8 – Zusammenfassung

Dieses letzte Kapitel gibt einen Überblick über den Gegenstand der vorliegenden Arbeit sowie alle erzielten wesentlichen Ergebnisse. Weiterhin werden Ideen zu deren Verbesserung und Weiterführung geäußert.

Alle Rechte vorbehalten

2. Hintergrund

Die vorliegende Diplomarbeit basiert auf einem vorhergehenden Projekt, welches von der TSI[1] in Zusammenarbeit mit der FHTW Berlin durchgeführt wurde. In den folgenden Abschnitten sollen zunächst dieses Projekt und der entstandene ESG-Client vorgestellt sowie daraus resultierende Vorgaben für diese Arbeit benannt werden. Soweit keine anderen Quellen benannt sind, basieren alle im Folgenden aufgeführten Informationen auf Kenntnissen, die im Rahmen des zugrunde liegenden Projektes erworben wurden.

2.1. DVB-H-Projekt

Unterwegs Fernsehen auf dem Laptop ist heute keine Zukunftsvision mehr dank PC-Cards[2], die als Empfänger für DVB-T-Signale fungieren. *Digital Video Broadcasting* (DVB)[3] steht für Digitales Fernsehen, das über Satellit (DVB-S), über Kabel (DVB-C) oder über terrestrische Antennen (DVB-T und in Zukunft auch DVB-H) ausgestrahlt wird. Mit Hilfe von speziellen Empfangsgeräten sind Fernsehen oder Musik für jedermann empfangbar (vgl. [dvb]). Fernsehen auf mobilen Kleinstgeräten wie PDA's ist daher bereits möglich, jedoch ist DVB-T als Übertragungstechnologie aufgrund des hohen Energieverbrauchs für diese kleinen Mobilcomputer ungeeignet (vgl. [frisch], S. 2), da die Daten in einem konstanten Datenstrom (engl. Stream) übertragen werden und der Empfänger immer eingeschaltet bleiben muss. Aus diesem Grund wird die ohnehin schon kurze Akkulaufzeit der kleinen Geräte weiter reduziert. Ein anderer Grund, der die Verwendung dieser Technologie für mobilen Empfang unmöglich macht ist die Problematik, dass eine Übertragung bei hohen Bewegungsgeschwindigkeiten des Empfangsgerätes, z.B. im Auto, nicht funktioniert.

Um diese Probleme zu lösen wurde vom DVB-Projekt die neue Technologie DVB-H[4] geschaffen. Diese stellt einen Übertragungsstandard speziell für kleine mobile Endgeräte dar, bei dem das Hauptaugenmerk auf einem möglichst geringen Energieverbrauch liegt. Statt der kontinuierlichen Übertragung der Daten wird bei DVB-H ein anderes Verfahren angewandt, das so genannte *IP Datacast*[5]. Die Daten werden dabei in kleinen IP-Paketen in einzelnen „Bursts“ stoßweise versendet. Dadurch ist es möglich eine unbegrenzte Menge an Daten

1 Siehe http://www.t-systems-mediabroadcast.de/coremedia/generator/www.t-systems-mediabroadcast.com/de/Home/templateId=renderNormal/id=46516.html
2 Siehe http://www.pcmcia.org/pccard.htm
3 Siehe http://www.dvb.org/
4 Siehe http://www.dvb.org/documents/white-papers/wp07.DVB-H.final.pdf
5 Siehe http://www.ipdc-forum.org/

Alle Rechte vorbehalten

beliebiger Art zu übertragen. Das Empfangsmodul kann bei Anwendung dieses Verfahrens zwischenzeitlich abgeschaltet und somit Strom gespart werden (vgl. [etsi1], S. 5).

Zugang zum digitalen Fernsehen erhält der Benutzer[6] über Empfangsgeräte, auf denen eine entsprechende Software installiert ist. DVB-T-Signale werden über gewöhnliche Antennen von Set-Top-Boxen empfangen, verarbeitet und auf dem Fernseher präsentiert. Bei dem übertragenen Datenstrom handelt es sich um einen *MPEG-2 Transport Stream* (MPEG-2 TS). Da die Daten in komprimierter Form (MPEG-2) versendet werden, kann ein solcher Datenstrom bis zu vier Fernsehprogramme enthalten (vgl. [frisch], S. 2). Dies stellt einen erheblichen Vorteil zur analogen Fernsehübertragung dar, bei der pro Frequenz lediglich ein Programm gesendet werden kann. Neben den audio-visuellen Inhalten sind in jedem Stream u.a. *Service-Informationen* (DVB-SI)[7] enthalten, die in Form von Tabellen den Aufbau und Inhalt des jeweiligen Streams beschreiben. Diese Informationen bilden die Grundlage für den Teletext und den so genannten *Electronic Program Guide* (EPG) (vgl. [digi], S. 15, 78).

Der EPG stellt eine Programmzeitschrift in elektronischer Form dar und ist standardmäßig als Software auf DVB-T-Receivern installiert. Sinn und Zweck eines EPG ist es, eine vom Benutzer erzeugte Programmdatenbank zu speichern und somit dem Benutzer die gewünschten Programme sowie Informationen über diese Programme schnellstmöglich zugänglich zu machen (vgl. [etsi2], S. 10). Die vom Programmanbieter ausgesendeten Service-Informationen werden vom EPG aus dem Datenstrom ausgelesen und dem Benutzer in übersichtlicher Form präsentiert. Je nach Anbieter sind verschiedene Informationen zu den aktuell laufenden und folgenden Sendungen verfügbar, z.B. der Titel einer Sendung, Start- und Endzeit oder eine kurze Beschreibung des Inhalts (vgl. [epg]).

Während es sich bei DVB-T vorwiegend um die Übertragung von Fernsehprogrammen handelt, ermöglicht das IP Datacast-Verfahren über DVB-H die Versendung von Daten jeglicher Art (vgl. [ipdata1]). Aufgrund dieser Vielfalt werden die übertragenen Inhalte auch *Services* genannt. Zum Empfang von DVB-H-Signalen sind eine Mobilfunkantenne und ein entsprechender Receiver notwendig. Beim Einschalten des Empfängers wird mit den DVB-SI-Tabellen zunächst eine IP-Adresse übertragen, unter welcher der so genannte *Electronic Service Guide* (ESG) zu finden ist. Der ESG ist ein Protokoll, das Informationen über alle ausgesendeten Services enthält. Zugang zu diesen Diensten erhält der Benutzer über einen *ESG-Client*. Diese Software liest die spezifischen Informationen über die übertragenen

6 Unter dem Begriff Benutzer sind in der vorliegenden Arbeit sowohl männliche als auch weibliche Personen zu verstehen. Ein Benutzer steht stellvertretend für einen beliebigen Menschen bzw. eine beliebige Person.
7 Siehe http://webapp.etsi.org/exchangefolder/en_300468v010601p.pdf

Alle Rechte vorbehalten

Services aus dem ESG aus und stellt diese übersichtlich dar, so dass der Benutzer einen Überblick über das verfügbare Angebot an Services bekommt. Wird ein bestimmter Service ausgewählt, erhält der Receiver eine IP-Adresse unter der die jeweiligen IP-Pakete zu diesem Service empfangen werden können. Die IP-Pakete werden entpackt und der gewählte Dienst über eine externe Applikation, z.B. einen Mediaplayer oder Browser, gestartet (vgl. [ipdata2]). Der Aufbau eines DVB-H-Systems wird in Abbildung 1 veranschaulicht. Ausführliche Informationen und Erläuterungen zum genauen Aufbau und der Funktionsweise geben [etsi1] und [etsi3].

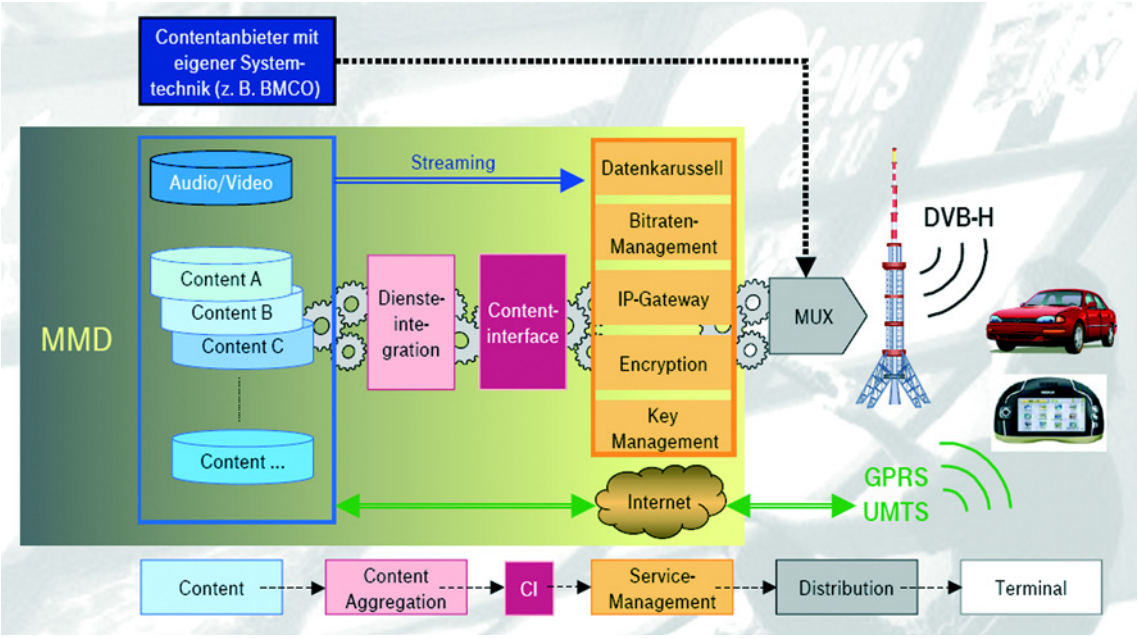

Abbildung 1: Aufbau des DVB-H-Systems ([frisch], S. 14)

Im Rahmen weltweiter Feldtests wurde von der TSI in Berlin ein DVB-T-Kanal für den Praxistest des DVB-H-Standards freigegeben. Daten werden bei diesem Feldtest über DVB-H versendet und über mobile Geräte mit DVB-H-Receivern empfangen. An einer Implementierung für solche Empfangsmodule arbeiten bereits verschiedene Firmen zusammen. Die im Vergleich zum Desktop-Computer geringe Prozessorleistung der kleinen mobilen Endgeräte erlaubt nur niedrige Kompressionsraten des Videomaterials. Daher sollte diese optimal ausgenutzt und die ohnehin schon kurze Betriebsdauer nicht noch weiter verringert werden. Somit kommen momentan keine schnelleren Prozessoren in Frage. Aufgrund dieser Problematik werden u.a. im Heinrich-Hertz-Institut ein neuer Codec, *H.264* (eine MPEG-4-Erweiterung), und ein dazugehöriger Player entwickelt. Dies ermöglicht eine effiziente Codierung des Videomaterials sowie die trotz allem noch sehr rechenintensive Dekodierung mit den gegenwärtig verfügbaren PDA-Prozessoren. Bis zur Fertigstellung der erforderlichen Technologien wurden, so auch im Rahmen des Projektes, die Testdaten über

DVB-T versendet und von Computern mit entsprechender Hard- und Software empfangen. Ziel der TSI ist es, bis zur Fußball-Weltmeisterschaft 2006 DVB-H zur Übertragung von Spielsequenzen an eine Vielzahl von mobilen Empfangsgeräten einzusetzen.

Im Rahmen eines Studienprojektes, welches die TSI gemeinsam mit Studenten der FHTW Berlin durchgeführt hat, ist ein Prototyp eines ESG-Client entstanden. Ziel dieses Projektes war es, eine funktionierende Anwendung zu schaffen, welche die empfangenen Testdaten auslesen und Informationen über das verfügbare Angebot übersichtlich aufbereitet darstellen kann. Dieser Prototyp wird gegenwärtig in einem Anschlussprojekt weiterentwickelt und soll im September 2005 auf der Internationalen Funkausstellung in Berlin präsentiert werden.

Abbildung 2: OnAir-Darstellung des ESG-Client-Prototypen

(1 – Content-Anbieter, 2 – Sendezeit, 3 – Kurzbeschreibung, 4 – verfügbare Dienste, 5 – Titel des Dienstes, 6 – Tableiste, 7 – Start des Dienstes, 8 – Hinweis auf Kostenpflicht)

Ergebnis des Pilot-Projektes waren ein prinzipiell[8] funktionstüchtiger ESG-Client, bestehend aus einzelnen Plug-Ins für verschiedene Funktionen, sowie eine Zwischenschicht, der *DVB-H Abstraction Layer* (DVB-H AL), über welche der ESG-Client mit dem unterliegenden Datenmodell der TSI kommunizieren und die verfügbaren Services auslesen kann. Zu den implementierten Funktionen gehören eine Programmübersicht in verschiedenen Rubriken, die Möglichkeit der Auflistung aller aktuell angebotenen Dienste in einer „OnAir"-Darstellung (siehe Abbildung 2), eine Suche nach bestimmten Diensten in vorgegebenen Kategorien oder

8 Der ESG-Client funktionierte mit Testdaten, die direkt in die Zwischenschicht DVB-H AL eingegeben wurden, konnte jedoch im Rahmen des Projektes nicht zusammen mit dem unterliegenden Modell der TSI getestet werden, da dieses noch nicht voll funktionsfähig war. Auch eine Übertragung über DVB-H war noch nicht möglich, da der Testkanal noch nicht freigegeben war und kein Empfangsmodul zur Verfügung stand.

Alle Rechte vorbehalten

durch beliebige Texteingabe sowie die Möglichkeit, ähnlich wie beim Fernsehen durch alle verfügbaren TV-, Audio- oder sonstigen Service-Dienste zu „zappen“.

Die Bedienung der Anwendung erfolgt auf dem PDA mit dem Steuerkreuz, dem Eingabestift oder dem Finger. Optimiert ist die Darstellung für das Querformat. Die Karteikartenmetapher am rechten bzw. linken Rand (abhängig von der Einstellung des Querformates als Rechts- oder Linkshänder) ermöglicht eine einhändige Bedienung des ESG-Clients z.B. in Bus und Bahn. Services sind für den Benutzer ausschließlich über diese Software zugänglich. Wird ein bestimmter Service gewählt, so wird dieser vom ESG-Client als externe Applikation gestartet. Der ESG-Client verbleibt im Hintergrund bis er durch den Benutzer beendet wird. Der Aufbau und die Funktionsweise des ESG-Client werden nachfolgend genauer betrachtet.

2.2. Vorstellung des Prototypen

Aufgrund der Größe des Projektteams sowie der verschiedenen Standorte der beiden Gruppen wurde der prototypische ESG-Client verteilt entwickelt. Um eine klare Trennung der Aufgaben zu gewährleisten wurde das MVC-Design-Konzept angewendet. Das Team der TSI arbeitete zu dem Zeitpunkt bereits an dem unterliegenden Datenmodell. Die Projektgruppe der FHTW erarbeitete daraufhin ein Konzept zur Erstellung der grafischen Benutzungsschnittstelle und deren Funktionsweise. Das Datenmodell der TSI wurde an die benötigten Funktionen angepasst sowie in Zusammenarbeit beider Gruppen eine Kommunikationsschicht entwickelt, über welche die grafische Benutzungsschnittstelle auf die unterliegenden Daten zugreifen kann, um diese darzustellen. Der im Rahmen des Projektes entstandene Prototyp eines ESG-Client lässt sich daher in drei Bereiche unterteilen (siehe Abbildung 3):

- die grafische Benutzungsschnittstelle mit Ereignisbehandlung (View/Controller)
- die Zwischenschicht DVB-H AL (Controller/Model)
- das Datenmodell (Model).

Bei dem durch TSI bereitgestellten Datenmodell handelt es sich um Objekte, die aus den mittels DVB-H übertragenen Daten generiert wurden. Diese werden durch die Ansichtsebene visualisiert. Über den Controller kann der Benutzer auf den ESG-Client einwirken und sowohl die Daten als auch deren Darstellung verändern (vgl. [kauf], S. 1).

Alle Rechte vorbehalten

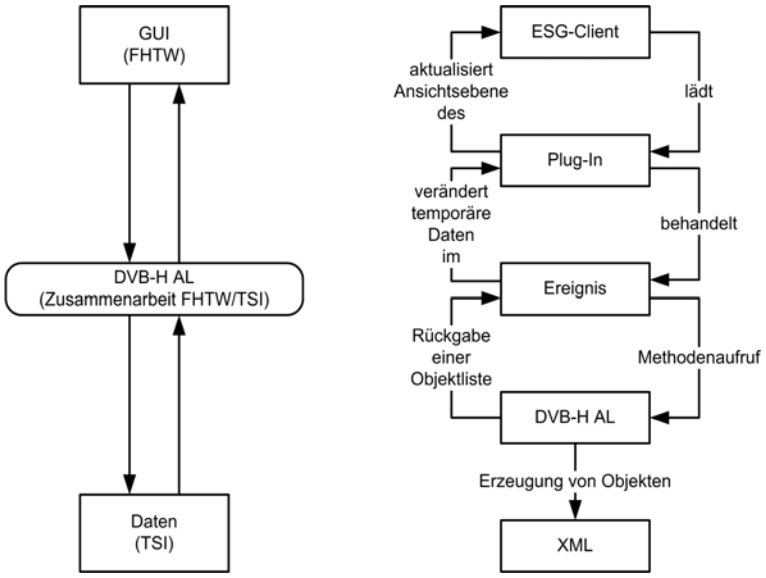

Abbildung 3: Zusammenspiel der einzelnen Komponenten des ESG-Client

Neben der Trennung nach MVC wurde zusätzlich die grafische Benutzungsschnittstelle in einzelne Komponenten – den ESG-Client selbst, einen Startbildschirm sowie mehrere *Plug-Ins* – unterteilt. Als Plug-Ins werden ausschließlich Klassenbibliotheken (engl. Assemblies, Dateiendung: „*.dll“) erkannt, die eine bestimmte Schnittstelle implementieren. Bei den Plug-Ins handelt es sich um Elemente, die vom Client über die genannte Schnittstelle aufgerufen, eingebunden und schließlich dargestellt werden. Beim Starten des ESG-Client lädt dieser den Startscreen sowie alle verfügbaren Plug-Ins. Über den Startscreen kann der Benutzer die Plug-Ins direkt ansteuern. Diese werden dann im Client angezeigt. Der Client selbst besitzt lediglich eine Ladefunktion, alle weiteren Funktionen werden von den einzelnen Plug-Ins bereitgestellt. Im Client werden die Plug-Ins in Form von *Tabs* (engl. für „Reiter“) repräsentiert. Beim Anklicken eines Tabs wird das entsprechende Plug-In angezeigt. Plug-Ins wurden verwendet, damit die Funktionalität des ESG-Client variierbar und problemlos erweiterbar ist, indem die einzelnen Plug-Ins beliebig vom Programmierer ausgetauscht oder vollständig entfernt werden können. Ebenso bot sich die Verwendung eines Plug-In-Konzeptes für die Entwicklung im Team an, da verschiedene Funktionen unabhängig von anderen Teammitgliedern implementiert werden konnten.

Die von einem Plug-In dargestellten Daten liegen zunächst in Form von XML-Dokumenten vor, die aus den übertragenen IP-Paketen erzeugt wurden. Um diese Daten visualisieren zu können, erfolgt im Plug-In der Aufruf einer Methode aus dem DVB-H AL. Die Abstraktionsschicht generiert daraufhin Listen von Objekten aus den vorliegenden XML-Daten. Dabei werden je nach Inhalt der IP-Pakete die ausgestrahlten Services in so genannte Video-, Audio- und Service-Items unterteilt sowie Hierarchien dieser Objekte erzeugt. Die

Alle Rechte vorbehalten

verfügbaren Items werden schließlich an das aufrufende Plug-In übergeben, woraufhin dieses wiederum die zur Veranschaulichung benötigten Daten, z.B. Name, Typ oder Anbieter des jeweiligen Services, ausliest und diese Daten in übersichtlicher Form für den Benutzer darstellt. Einen Überblick über die Kommunikation zwischen den einzelnen Komponenten gibt Abbildung 3.

Da sich diese Arbeit mit der Weiterentwicklung dieses Resultates einer Projektarbeit befasst, gelten die technischen Vorgaben, die bereits im Rahmen des zugrunde liegenden Projektes festgelegt wurden. Diese Vorgaben werden im folgenden Abschnitt erläutert.

2.3. Technische Vorgaben

In den technischen Vorgaben, die im Rahmen des DVB-H-Projektes festgelegt wurden, werden die Zielplattformen für den ESG-Client, daraus resultierend die zu verwendende Entwicklungsumgebung sowie auch die anzuwendenden Programmierkonzepte definiert.

Die entstehende Anwendung sollte sowohl auf Handheld-Terminals mit Windows Mobile 2003 Second Edition (SE) for Pocket PC als auch auf herkömmlichen Windows 32-Betriebssystemen einsatzfähig sein. Als Zielgeräte wurden Pocket PC, Tablet PC und Desktop PC festgelegt. Das Hauptaugenmerk für die Weiterentwicklung des prototypischen ESG-Client liegt jedoch auf der Plattform Pocket PC, insbesondere einem *Pocket PC 2003 SE*, da DVB-H speziell für die Benutzung auf kleinen mobilen Endgeräten entwickelt wurde. In der vorliegenden Arbeit ist unter dem Begriff Pocket PC 2003 SE grundsätzlich ein mobiler Kleinstcomputer zu verstehen, auf dem das Betriebssystem *Windows Mobile 2003 SE for Pocket PC* installiert ist. Für die Wahl dieses Betriebssystems sprechen v.a. das für eine optimale Videowiedergabe geeignete Querformat und die Unterstützung für Displays mit VGA-Auflösung. Der ESG-Client ist für die Verwendung im Querformat programmiert.

Da der Prototyp für die Verwendung auf Windows-basierten Plattformen ausgerichtet ist, muss die Entwicklungsumgebung Visual Studio .NET 2003 zur Programmierung benutzt werden. Mit Hilfe dieser Software ist es möglich, Applikationen für Betriebssysteme zu erstellen, die das .NET Framework bzw. die reduzierte Variante, das .NET Compact Framework, implementieren (siehe Kapitel 3.2.). Daher kann der ESG-Client einmal entwickelt und für die Verwendung auf verschiedenen Zielplattformen veröffentlicht werden. Die bestehende Anwendung ist vollständig in C#, einer speziell für das .NET Framework geschaffenen Programmiersprache, implementiert (vgl. [csharp], S. 3). Zum Testen der

Alle Rechte vorbehalten

Anwendung während der Implementierungsphase wird ein Pocket PC-Emulator verwendet. Die dazu nötige Software, die Emulator Images for Pocket PC 2003 SE[9], lässt sich problemlos in die Entwicklungsumgebung integrieren. Der Emulator verhält sich wie ein richtiger Pocket PC und ermöglicht Funktionstests der Anwendung direkt aus der Entwicklungsumgebung heraus, ohne das physische Vorhandensein des Zielgerätes. Das umständliche und zeitaufwändige Kopieren des Codes sowie die Anschaffung eines teuren Endgerätes sind daher zur Implementierung nicht notwendig. Im Rahmen des Projektes wurde die englische Sprache als Bedienungssprache festgelegt. Daher werden zum Testen der Implementierung die englischen Emulator Images benötigt. Aus diesem Grund soll der WWW PPC 2003 SE Landscape SDK-Emulator verwendet werden (Querformat, englische Sprache). Die Dokumentation des Programmcodes erfolgte während des Projektes zum größtem Teil auf Deutsch und wird demzufolge in deutscher Sprache fortgeführt.

Aufgrund der hohen Anforderungen, welche die DVB-H-Technologie an das mobile Endgerät stellt, konnte ausschließlich das beste und schnellste Gerät auf dem Markt, zum gegenwärtigen Zeitpunkt der Dell Axim x50v[10], als Entwicklungsgrundlage gewählt werden. Dieser bietet eine VGA-Auflösung und eine Prozessorleistung von 624MHz, welche das momentane Maximum bei PDA-Prozessoren darstellt. Der integrierte Grafikbeschleuniger ermöglicht DVD-Qualität beim Abspielen von Videomaterial. Weiterhin ist das geforderte Betriebssystem Windows Mobile 2003 SE for Pocket PC installiert und somit die Videowiedergabe und Verwendung des ESG-Client im Querformat möglich. Im CompactFlash-Steckplatz für entsprechende Speicherkarten könnte in naher Zukunft der DVB-H-Empfänger eingesteckt werden.

Bei der Weiterentwicklung des ESG-Client sollen die zugrunde liegende System-architektur sowie das Design der grafischen Benutzungsschnittstelle der Anwendung möglichst erhalten bleiben. Die Vorgaben hinsichtlich der Zielplattformen sowie der Entwicklungsumgebung sind zu befolgen und bilden zusammen mit der aus dem Projekt resultierenden Implementierung des ESG-Client-Prototypen die Basis für diese Diplomarbeit.

Im folgenden Kapitel sollen die zum Verständnis der Arbeit erforderlichen grundlegenden Kenntnisse vermittelt werden. Dabei handelt es sich einerseits um technische Details zum zuvor beschriebenen Projekthintergrund sowie andererseits um die Vorstellung neuer Technologien, die im Rahmen dieser Arbeit verwendet werden sollen.

9 Siehe http://www.microsoft.com/downloads/details.aspx?FamilyID=5c53e3b5-f2a2-47d7-a41d-825fd68ebb6c&displaylang=en

10 Siehe http://www1.us.dell.com/content/products/productdetails.aspx/axim_x50v?c=us&cs=19&l=en&s=dhs#tabtop

Alle Rechte vorbehalten

3. Grundlagen

Die aus dem DVB-H-Projekt hervorgehenden technischen Grundlagen dieser Arbeit werden in den folgenden Abschnitten genauer betrachtet sowie zusätzliche, zur Weiterentwicklung des ESG-Client benötigte Technologien vorgestellt.

3.1. Begriffserklärung

In diesem Kapitel sollen zunächst die in der Arbeit verwendete Terminologie erklärt und Abgrenzungen zu verwandten Begriffen geschaffen werden. Die Begriffe Skinning, Branding und Theme sind in der Informationstechnologie im Rahmen der Softwareentwicklung, insbesondere auf dem Gebiet Design, häufig anzutreffen. In dieser Arbeit ist das Verständnis dieser Worte von Bedeutung, da die Grundlage zur Neugestaltung der Benutzungsschnittstelle eines Programms geschaffen werden soll.

Skinning

Unter dem Begriff *Skinning* ist in der vorliegenden Diplomarbeit die Modifikation der grafischen Benutzungsschnittstelle eines Programms zu verstehen. Das bedeutet, dass der für den Benutzer sichtbare Teil der Anwendung ein neues Aussehen erhält. Das jeweilige Aussehen wird dabei auch als ein *Skin* (engl. für Haut) bezeichnet. Für ein skinningfähiges Programm können beliebig viele solcher Skins definiert werden. In einem Skin werden dabei u.a. die Farben, in denen bestimmte Elemente angezeigt werden sollen, oder auch Größen und Positionen für bestimmte Elemente festgelegt. Im Allgemeinen kann in den Optionen des Programms der zu verwendende Skin ausgewählt und somit das Aussehen der Benutzungsschnittstelle nach Belieben verändert werden. Besteht diese Möglichkeit, so ist es zumeist für den Benutzer ebenso möglich, eigene Skins zu erstellen. Im Rahmen dieser Arbeit werden Skins innerhalb von *Stylesheet-Dokumenten* (kurz: *Stylesheets*) abgelegt. Diese XML-Dokumente können beliebig viele Skins enthalten. Es können ebenso mehrere Stylesheets erzeugt werden, die wiederum verschiedene Skins enthalten. Einer Anwendung muss in diesem Fall jeweils ein Stylesheet und ein darin definierter Skin zugewiesen werden.

Theme

Wird bei der Gestaltung der einzelnen Skins ein bestimmtes Motiv verwendet, z.B. eine Jahreszeit, eine Sportart oder ein Kinofilm, welches sich in mehreren Elementen wiederfinden lässt, so werden die erstellten Skins auch als *Themes* (Themen) bezeichnet.

Alle Rechte vorbehalten

Branding

Branding steht im allgemeinen Sprachgebrauch für das Versehen eines Produktes mit einer Marke, durch die es unverwechselbar wird. Im Rahmen dieser Arbeit ist damit eine Form von Skinning gemeint, allerdings mit dem Unterschied, dass nicht der Benutzer das Aussehen der GUI verändert sondern eine Anwendung an verschiedene Firmen mit unterschiedlichem Aussehen verkauft wird. Dabei wird der Skin an das Corporate Design, u.a. Logos und Farben, der Firmen angepasst. Vertreiben die Firmen die Software an die Endkunden wird gewährleistet, dass diese das Programm als Produkt der jeweiligen Firma wiedererkennen.

3.2. Pocket PC

Mobilcomputer jeglicher Art werden unter der englischen Bezeichnung *Handheld* zusammengefasst. Zu diesen kleinen Geräten zählen u.a. Palmtops, Smartphones und Organizer. Eine weitere Kategorie stellen die Personal Digital Assistants dar, deren besondere Merkmale die Größe einer Handfläche und die fehlende Tastatur sind (vgl. [hand]). Eine typische Erscheinungsform eines PDA ist der Pocket PC, der im Folgenden näher betrachtet werden soll.

Pocket PC (kurz: PPC) ist sowohl ein Betriebssystem als auch eine Hardware-Plattform. Ist Ersteres gemeint, so sind darunter alle Windows CE-Betriebssystemvarianten ab Version 3.0 zu verstehen. Ist das Gerät gemeint, werden damit kleine Mobilcomputer bezeichnet, die mit einem Touchscreen und einem Stift zur Kommunikation mit dem Benutzer sowie dem Betriebssystem Pocket PC und zahlreichen Funktionen wie Terminplaner, Text-verarbeitungsprogramm, Internetbrowser und E-Mail-Programm ausgestattet sind. Wie in Kapitel 2.3. bereits erwähnt, stellt ein Pocket PC 2003 SE die Grundlage für die weiter zu entwickelnde Implementierung eines ESG-Client dar. Aufgrund dessen soll in diesem Kapitel eine Abgrenzung zu anderen PPC-Versionen getroffen und genauer auf die bei der Entwicklung zu beachtenden Besonderheiten des Pocket PC 2003 SE eingegangen werden. Eine Übersicht über alle Pocket PC-Plattformen und -Betriebssysteme gibt Tabelle 1:

Tabelle 1: Pocket PC-Übersicht (vgl. [wince1] und [wince2])

Name	***Basis***	***Betriebssystem-Name***
Pocket PC	Windows CE 3.0	Pocket PC 2000
Pocket PC 2002	Windows CE 3.0 (Version 11171) + zusätzliche Funktionen	Pocket PC 2002

Alle Rechte vorbehalten

Name	*Basis*	*Betriebssystem-Name*
Pocket PC 2003	Windows CE .NET (Version 4.2)	Windows Mobile 2003 for Pocket PC
Pocket PC 2003 Second Edition	Windows CE .NET (Version 4.21) + zusätzliche Funktionen	Windows Mobile 2003 Second Edition for Pocket PC

Wie in Tabelle 1 zu erkennen, stellt Windows Mobile 2003 SE for Pocket PC das installierte Betriebssystem der Plattform Pocket PC 2003 SE dar. Den Kern dieses Betriebssystems bildet Windows CE .NET. Windows CE ist eine Windows-Betriebssystemvariante speziell für kleine mobile Endgeräte (vgl. [wince1]). Das Kürzel .NET weist darauf hin, dass das .NET Framework in dieser Windows CE-Version implementiert ist. Dieses Framework ist das Programmiermodell von .NET (vgl. [west], S. 116). Es setzt sich zusammen aus einer allen Programmiersprachen *gemeinsamen Laufzeitumgebung* (CLR, Common Language Runtime) und einer *Basisklassenbibliothek*[11] (FCL, Framework Class Library) und soll die Entwicklung und Veröffentlichung moderner, komponentenbasierter Anwendungen erleichtern (vgl. [csharp], S. 3). Ein mögliche Darstellung des .NET Frameworks ist in Abbildung 4 zu sehen. Es wird dabei in zwei Bereiche unterteilt, das Grundgerüst und einen darauf aufgesetzten Rahmen mit einer Vielzahl zusätzlicher Funktionen:

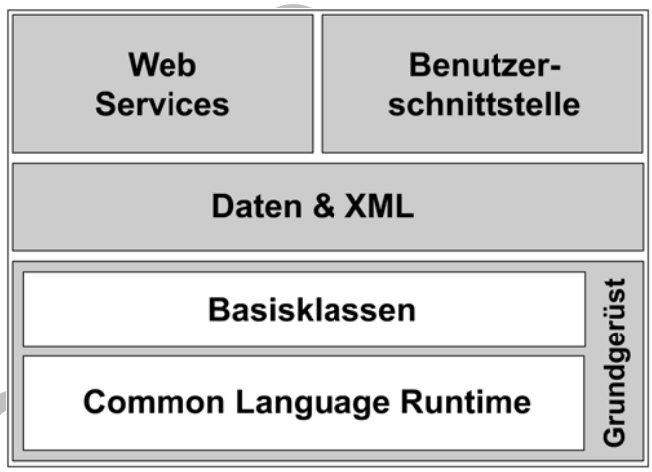

Abbildung 4: Aufbau des .NET Frameworks ([vasters], S. 8)

Aufgrund des geringeren Speicherplatzes wird auf den kleinen Mobilgeräten eine im Funktionsumfang reduzierte Version, das *.NET Compact Framework* (kurz: .NET CF), verwendet. „.NET Compact Framework ist eine hardwareunabhängige Umgebung für die Ausführung von Programmen auf Computergeräten mit beschränkten Ressourcen. [...] .NET

11 „Die .NET Framework-Klassenbibliothek ist eine Bibliothek aus Klassen, Schnittstellen und Werttypen, die in Microsoft .NET Framework SDK enthalten sind. Diese Bibliothek bietet Zugriff auf Systemfunktionen und wurde als Grundlage für .NET Framework-Anwendungen, -Komponenten und -Steuerelemente entworfen.“ [fcl]

Alle Rechte vorbehalten

Compact Framework ist eine Teilmenge der .NET Framework-Klassenbibliothek und enthält darüber hinaus Klassen, die ausschließlich für .NET Compact Framework entworfen wurden. Es nutzt die vollständige .NET Framework Architektur mit der Common Language Runtime und der verwalteten Codeausführung“ ([netcf]). Dies bedeutet, dass der Programmcode in einer beliebigen, von .NET unterstützten Sprache geschrieben und auf beliebigen Plattformen ausgeführt werden kann. Dabei kann der Programmcode für .NET CF-Anwendungen in den Programmiersprachen Visual Basic .NET oder Visual C# geschrieben werden, wobei die einzelnen Komponenten einer Anwendung in verschiedenen Sprachen vorliegen können (vgl. [richter], S. XXII f.). Bei der Kompilierung wird der Code zunächst in eine Zwischensprache umgewandelt und erst zur Laufzeit des Programms in den entsprechenden Maschinencode überführt (*Just-In-Time-Kompilierung*). Somit wird Plattformunabhängigkeit hinsichtlich der unterliegenden Hardware, dem Betriebssystem und der Programmiersprache ermöglicht (vgl. [vasters], S. 129 f.). Bei der Entwicklung einer Anwendung, die auf verschiedenen Plattformen ausführbar sein soll muss jedoch beachtet werden, dass ausschließlich Funktionen benutzt werden, die sowohl vom .NET Framework als auch vom .NET Compact Framework unterstützt werden.

Ein weiterer Aspekt, der bei der Erstellung einer Anwendung für Pocket PC 2003 SE beachtet werden sollte, ist die Umschaltmöglichkeit der Bildschirmanzeige zwischen dem Hoch- und Querformat. Beim Start der Applikation und beim Formatwechsel sollte zunächst getestet werden, welches Format eingestellt ist und somit die Anordnung der dargestellten Elemente entsprechend verändert werden. Soll die Anwendung auf unterschiedlichen Plattformen benutzt werden können, sollte generell das unterliegende Betriebssystem überprüft und dementsprechend die jeweilige Ausgabegröße festgelegt werden. Bei kleinen Mobilcomputern wie Pocket PC's ist weiterhin die geringe Bildschirmgröße zu bedenken. Dies erschwert zusätzlich die ansprechende Gestaltung der grafischen Benutzungs-schnittstelle. Die Bedeutung solcher Benutzungsschnittstellen sowie die Probleme bei deren Gestaltung sollen im folgenden Abschnitt erläutert werden.

3.3. Benutzungsschnittstellen

Um Computerprogramme benutzen zu können, ist eine Schnittstelle zwischen Mensch und Maschine notwendig, die es dem Benutzer ermöglicht den Programmablauf zu steuern, um das angestrebte Ergebnis zu erzielen. Die Mensch-Computer-Interaktion erfolgt durch die Kommunikation über eine so genannte *Benutzungsschnittstelle* (UI, User Interface). Dabei

Alle Rechte vorbehalten

tätigt der menschliche Benutzer Eingaben am Computer über Geräte, wie beispielsweise eine Tastatur, eine Maus, einen Touchscreen oder aber auch über ein Mikrofon. Die Reaktion des Systems erfolgt nach Verarbeitung dieser Informationen mittels entsprechender Ausgaben u.a. auf einem Bildschirm oder Drucker. Eine Benutzungsschnittstelle lässt sich aufgrund dieses Handlungsablaufes in drei Bereiche gliedern (vgl. [hein], S. 29 f.):

- Ein-/Ausgabe (Eingabe bzw. Darstellung von Informationen)
- Werkzeuge (Vom Programm angebotene Funktionen zu dessen Steuerung)
- Dialoge (Angabe der erforderlichen Informationen, Aufruf der nötigen Funktionen).

Die in der Anwendung verwendete Dialogart legt anhand der zur Interaktion benötigten Eingabegeräte fest, welche Art von Benutzungsschnittstelle vorliegt. So werden beispielsweise kommandozeilenbasierte und menügesteuerte Benutzungsschnittstellen unterschieden. Vorherrschend sind heute *grafische Benutzungsschnittstellen* wie sie in Windows- oder Macintosh-Betriebssystemen vorzufinden sind. Diese verwenden Dialoge mit Fenstersystemen (vgl. [hein], S. 154). Ein Beispiel dafür gibt Abbildung 5:

Abbildung 5: Dialoge mit Fenstersystemen

(Fensterelemente: 1 – Titelbalken mit Anwendungsmenü-Knopf, 2 – Menübalken, 3 – Arbeitsbereich, 4 – selektiertes Objekt bzw. Piktogramm, 5 – Schließen-Knopf, 6 – abgeblendeter Vollbild-Knopf, 7 – Minimieren-Knopf, 8 – Rollbalken, 9 – Rahmen)

„*Fenster* sind abgegrenzte rechteckige Bereiche des Bildschirms, die zur Darstellung und Manipulation von Objekten einer Anwendung benutzt werden“ ([hein], S. 160). Abbildung 5 zeigt zwei unterschiedliche Ausprägungen von Fenstersystemen. In einem Anwendungsfenster des Betriebssystems Windows XP läuft die Anwendung Pocket PC-Emulator, in der wiederum ein Fenster des Betriebssystems Windows Mobile 2003 SE geöffnet ist. Wird das

Anwendungsfenster geschlossen, so wird gleichzeitig die darin ausgeführte Anwendung, der Pocket PC-Emulator, beendet. Beide Fenster weisen typische Fensterelemente, wie einen Titelbalken und einen Schließen-Knopf (siehe Bildunterschrift 1-9), auf, wobei abgesehen vom Arbeitsbereich das Auftreten aller weiteren Elemente optional ist (vgl. [hein] S. 161).

Beim Arbeitsbereich eines Fenstersystems handelt es sich um ein *Formular*. „Formulare sind Gruppierungen von Interaktionselementen, die zur Beschreibung einer bestimmten Teilaufgabe angezeigt werden“ ([hein], S. 158). Abhängig von der Art der Anwendung kann der Arbeitsbereich beispielsweise ein Formular zur Dateneingabe, Datenbearbeitung oder Datenanzeige sein. Die in einem Formular dargestellten Elemente sowie die zuvor genannten Fensterelemente gehören zur Gruppe der Interaktionselemente aus dem Bereich der Ein- und Ausgabe.

Bei grafischen Benutzungsschnittstellen werden im Bereich der visuellen Informationsdarstellung *Interaktionselemente* zusätzlich zu textuellen Informationen bereit gestellt. Der Benutzer kann mit Hilfe dieser Elemente Dateneingaben vornehmen und somit mit dem System interagieren. Neben den durch die Entwicklungsumgebung oder das Betriebssystem bereitgestellten Standard-Interaktionselementen können zusätzlich vom Programmierer applikationsabhängige Interaktionselemente geschaffen werden. Die Gruppe der Standard-Interaktionselemente umfasst Basiselemente, wie Eingabefelder, Schaltflächen, Auswahllisten oder Grafiken, und Erweiterungselemente, z.B. Rollbalken (vgl. [hein], S. 135).

Bei der Erstellung grafischer Benutzungsschnittstellen werden *Metaphern* verwendet. Eine Metapher ist „die Übertragung von Konzepten und Handlungsweisen aus einem Bereich der gegenständlichen Welt auf Interaktionen am Rechner“ ([hein], S. 163). Anwendung findet die Kombination von Dialogen mit Fenstersystemen und Metaphern beispielsweise in Windows-Betriebssystemen. Der Benutzer blickt dabei auf eine Schreibtischoberfläche, auf der wiederum Ordner und Dokumente, repräsentiert durch Piktogramme, abgelegt sind.

Der Grund für die Verwendung grafischer Benutzungsschnittstellen in Anwendungen ist neben Zeitersparnis, einem ansprechenderen Aussehen sowie einem besseren Überblick über alle verfügbaren Funktionen vor allem die einfache und schnelle Erlernbarkeit einer Anwendung. Eine intuitive und angstfreie Bedienung einer Anwendung wird durch die Verwendung von anklickbaren, selbsterklärenden Interaktionselementen und Metaphern gewährleistet. Der Benutzer kann sich durch die Übertragung von Kenntnissen aus seinem alltäglichen Umfeld leichter in einem neuen Programm zurecht finden ohne zuvor Hintergrundwissen erwerben zu müssen, wie z.B. eine Kommandosprache wie MS-DOS.

Alle Rechte vorbehalten

Zusätzlich zu den zuvor beschriebenen Hilfsmitteln zur Vereinfachung der Mensch-Computer-Interaktion gibt es Gestaltungsgrundsätze, mit deren Hilfe die Gestaltung der grafischen Benutzungsschnittstelle und deren Interaktionselementen erleichtert werden soll. Diese Normen wurden von der *International Organization for Standardization* (ISO)[12] festgelegt und beinhalten Vorgaben für Schriftgröße, Farbe, Anzahl und Anordnung von Elementen. Ziel ist dabei eine ergonomische Informationsdarstellung angepasst an die Eigenschaften der menschlichen Wahrnehmung. Diese Regelungen betreffen alle genannten Ebenen einer Benutzungsschnittstelle und sollen bewirken, dass die Informationen für den Benutzer leichter erfassbar und verständlicher sind (vgl. [hein], S. 131). An dieser Stelle sollen die ISO-Normen jedoch nicht genauer betrachtet werden, da die Gestaltung einer Benutzungsschnittstelle nicht Thema der vorliegenden Arbeit ist. Es soll lediglich eine existierende Benutzungsschnittstelle soweit verändert werden, dass eine einfache Neugestaltung dieser möglich ist. Die Gestaltung selbst wird jedoch von anderen Personen vorgenommen und liegt demzufolge außerhalb des Rahmens dieser Arbeit.

Es stellen jedoch nicht alle Benutzer die gleichen Ansprüche an die Benutzungsschnittstelle einer Anwendung, da jeder Mensch andere Fähigkeiten und Vorlieben besitzt. Findet die Gestaltung der grafischen Benutzungsschnittstelle eines Programms Anklang bei einer Vielzahl von Benutzern, so gibt es doch immer Ausnahmen. Daher sollte dieser Bereich möglichst flexibel sein und an verschiedene Benutzer individuell angepasst werden können. Ebenso wie .NET das Verwenden austauschbarer Komponenten ermöglicht, sollte auch die grafische Benutzungsschnittstelle eines Programms austauschbar sein. Problematisch ist dies jedoch, wenn die Definition dieser Ansichtsebene vollständig im Programmcode integriert ist, da bereits kleine Änderungen an der Darstellung große Änderungen im Programmcode erfordern können. Aus diesem Grund müssen verschiedene Ansätze gefunden werden, wie die Definition der grafischen Benutzungsschnittstelle eines Programms ausgelagert werden kann, so dass der Programmcode von Änderungen der Darstellung unberührt bleibt. Einige Lösungsansätze werden in den folgenden Abschnitten vorgestellt.

3.4. Auszeichnungssprachen

Der Begriff *Markup* (engl. für Textauszeichnung) hat seinen Ursprung in der Druckindustrie und bezeichnet handschriftliche Anweisungen eines Layouters, die das Seitenformat, die Anordnung der Inhalte sowie die Typographie der im Dokument enthaltenen Texte festlegen

12 Gremium zur Schaffung weltweit einheitlicher Standards, so genannten ISO-Normen
Siehe http://www.iso.org/iso/en/ISOOnline.frontpage

Alle Rechte vorbehalten

(vgl. [jen]). Heute werden mit Hilfe so genannter *Markup Languages* (engl. für Auszeichnungssprachen) im Bereich der Informationstechnologie Daten und/oder deren Darstellungsweise beschrieben. Es lassen sich dabei mehrere Arten von Auszeichnungen unterscheiden, z.B. Procedural Markup und Presentational Markup (vgl. [coombs], S.935).

Die vorliegende Diplomarbeit befasst sich mit *beschreibenden Auszeichnungssprachen* (engl. Descriptive Markup Languages). Diese basieren auf dem Konzept des *Generic Markup*[13] und werden dazu verwendet, Dokumente zu strukturieren und deren Inhalte zugleich mit ihrer jeweiligen Bedeutung und Formatierung zu versehen. Ein Dokument wird dabei in logische Komponenten (*Elemente*) gegliedert, indem einzelne Textbereiche mit Hilfe so genannter *Tags* (<Tag>) markiert werden. Jedes Element wird durch einen Start- und einen End-Tag begrenzt. Die Tags geben dabei Aufschluss über die Identität und Bedeutung eines Elementes: Dies ist ein Paragraph . Die auf diese Art und Weise beschriebenen Daten können daher von anderen Anwendungen gelesen und entsprechend verarbeitet, z.B. für die Ausgabe auf dem Bildschirm oder Drucker formatiert oder in andere Sprachen transformiert, werden. Zur Familie der beschreibenden Auszeichungssprachen gehören u.a. die *Hypertext Markup Language* (HTML), die *Extensible Hypertext Markup Language* (XHTML) und die *Scalable Vector Graphics* (SVG). Diese Sprachen wurden mit Hilfe so genannter *Metasprachen* geschaffen. Eine Metasprache ist eine „Sprache, mit der eine andere Sprache erklärt oder definiert wird, mit der also die Regeln (die Grammatik) für diese Sprache festgelegt werden“ [jen]. Beispiele für solche Metasprachen sind die *Standard Generalized Markup Language* (SGML) und die Extensible Markup Language.

SGML ist ein umfangreicher Standard (ISO 8879), der mit seinem Dokumenttyp HTML zur Auszeichnung von Webseiten den Weg in das Internet fand (vgl. [jen]). HTML befolgt das Konzept des Generic Markup jedoch nur zum Teil, da neben der Beschreibung der Bedeutung eines Textes, z.B. als Überschrift, auch die Darstellungsweise beschrieben wird, z.B. fett, unterstrichen oder farbig. HTML bietet zudem lediglich eine begrenzte Anzahl an fest vorgegebenen Tags, die zur Darstellung von Texten im Internet in Form von Webseiten gut geeignet, jedoch für die Anwendung in anderen Bereichen nicht ausreichend sind. Der eigentliche Vorteil der Flexibilität der beschreibenden Auszeichnungssprachen ist aufgrund der Komplexität von SGML und der begrenzten Möglichkeiten von HTML nicht gegeben. Um SGML flexibler im Bereich der Informationstechnologie einsetzen zu können wurde die Metasprache XML geschaffen (vgl. [jen]).

13 „Begriff für das Konzept der Auszeichnung von Inhalten nach semantischen anstatt nach visuellen Gesichtpunkten“ [jen]

Alle Rechte vorbehalten

3.4.1. XML

XML ist eine Teilmenge von SGML und somit kompakter, strenger und daher einfacher zu verwenden. Trotz seiner Strenge ist es flexibler einsetzbar, da im Gegensatz zu HTML keine vorgeschriebenen Tags existieren sondern diese, entsprechend der Bedeutung der Elementinhalte, beliebig definiert werden können. Ein einfaches XML-Dokument ist folgendermaßen aufgebaut:

```
 1| <?xml version="1.0"?>
 2| <!DOCTYPE autor SYSTEM "autor.dtd">
 3| <autor xmlns="http://www.merritkroeger.de">
 4|  <nachname>Kröger</nachname>
 5|  <vorname>Merrit</vorname>
 6|  <wohnort>Berlin</wohnort>
 7|  <geschlecht>weiblich</geschlecht>
 8|  <alter>23</alter>
 9|  <beruf>Studentin</beruf>
10| </autor>
```

Abbildung 6: Beispiel für ein einfaches XML-Dokument

Das XML-Dokument in Abbildung 6 ist wohlgeformt, da seine Syntax die vorgegebenen Regeln befolgt. Dies bedeutet, dass u.a. (vgl. [xml], S. 19):

- die zugrunde liegende XML-Version angegeben wird (siehe Zeile 1)
- jedes Element immer ein Start- und ein End-Tag besitzt
- Attribute (siehe Zeile 1) nicht mehrfach vorkommen und deren Werte in doppelten Anführungszeichen stehen
- alle Tags korrekt hierarchisch, ohne Überschneidungen verschachtelt sind.

Folgt das Dokument zusätzlich einer *Document Type Definition* (DTD, in Zeile 2 wird eine externe DTD eingebunden), welche die Definition der Elemente und Attribute sowie deren Beziehung zueinander vorgibt, so ist dieses Dokument gültig (vgl. [xml], S. 9). Entsprechend dem Konzept des Generic Markup werden in XML-Dokumenten keine Ausgabeformatierungen festgelegt. Die Darstellung der XML-Dokumente wird entweder den Anwendungen überlassen, die diese verarbeiten, oder explizit in einem Stylesheet definiert. Damit alle Elemente in einem XML-Dokument eindeutig sind bietet die XML-Spezifikation die Möglichkeit zur Verwendung von *Namensräumen* (Namespaces). Dazu kann für das gesamte Dokument oder für einzelne Elemente ein *Uniform Resource Identifier* (URI) angegeben werden (siehe Zeile 3) (vgl. [xml], S 11 ff.).

Damit eine Anwendung mit den in einem XML-Dokument enthaltenen Informationen arbeiten kann, wird dieses von einem Parser gelesen. Ein solcher Parser wird beispielsweise

Alle Rechte vc .alten

durch das *Document Object Model* (DOM) bereitgestellt. DOM, ebenso wie auch XML eine Empfehlung des *World Wide Web Consortiums* (W3C)[14], ist eine plattform- und sprachunabhängige Schnittstelle, die es Programmen ermöglicht auf Inhalt, Struktur und Layout eines HTML- oder XML-Dokumentes dynamisch zuzugreifen und diese zu verändern (vgl. [dom1]). DOM definiert sowohl die logische Struktur eines Dokumentes als auch wie darauf zugegriffen und dieses Dokument manipuliert werden kann. Das Dokument wird dabei im Speicher in Form einer Baumstruktur abgebildet, in der jedes XML-Element einen Knoten und jeder Knoten ein Objekt darstellt, mit dem wiederum in der Anwendung gearbeitet werden kann (vgl. [dom2]).

Aufgrund ihrer Erweiterbarkeit und damit einhergehenden Flexibilität ist die Extensible Markup Language vielseitig einsetzbar. Da es keine festgelegten Tags gibt, die sich bei Änderungen in der Spezifikation ebenso ändern, müssen XML-Dokumente nicht zwingend „gewartet“ und an Neuerungen angepasst werden. Sie können in dem Sinne nicht veralten und Daten daher in Form von XML-Dokumenten gespeichert werden. Zudem ist XML die Lösung zum Problem der Inkompatibilität zwischen Dokumenten und Systemen, da die Verwendung von beschreibendem Markup eine 1:1-Abbildung zwischen logischen Elementen und Markup garantiert. Die daraus resultierende Plattformunabhängigkeit erlaubt einen problemlosen Datenaustausch in Form von XML-Dokumenten zwischen unterschiedlichen Systemen (vgl. [coombs], S. 939 f.). Web Services und das .NET Framework basieren auf XML. Dokumenttypen wie XHTML und SVG werden mit Hilfe von XML definiert. Auch die Abstraktion von Benutzungsschnittstellen erfolgt bereits mit Hilfe von XML. Vom Mozilla-Projekt wurde die XML User Interface Language definiert, mit der die grafischen Benutzungsschnittstellen von Mozilla-Anwendungen sowie deren Skins beschrieben werden. Einen ähnlichen Ansatz verfolgt Microsoft mit der Extensible Application Markup Language, deren Erläuterung Gegenstand des folgenden Abschnitts ist.

3.4.2. XAML

Das voraussichtlich Ende 2006 erscheinende Windows-Betriebssystem mit dem Code-Namen „*Longhorn*“[15] bringt ein vollkommen neues Programmiermodell mit sich. Dieses setzt sich zusammen aus den folgenden Komponenten (vgl. [long1]):

14 Organisation zur Entwicklung von Web-Standards
Siehe http://www.w3.org/

15 Seit dem 24.05.2005 ist eine Beta-Version der Bestandteile Indigo und Avalon für Windows XP und Windows Server 2003 verfügbar.
Siehe http://www.microsoft.com/downloads/details.aspx?familyid=b789bc8d-4f25-4823-b6aa-c5edf432d0c1&displaylang=en

Alle Rechte vorbehalten

- Longhorn SDK (Klassenbibliothek zur Anwendungsentwicklung)
- WinFS (Dateisystem zur Speicherung von Dateien und deren Metadaten in einer Datenbank)
- Indigo (Technologien zur Entwicklung verteilter Anwendungen)
- Yukon (Implementierung von SQL-Server)
- Avalon (Teil des Longhorn SDK zur Entwicklung von Benutzungsschnittstellen)
- XAML (XML-basierte Scriptsprache zur Beschreibung von Benutzungsschnittstellen in Longhorn-Anwendungen).

Die Entwicklung von Anwendungen beruht auf dem „build-once, deploy n-times application model“ [long2]. Dies bedeutet, dass für Entwickler die bisher zwingende Entscheidung, ob eine Web-Anwendung oder eine Windows-Anwendung entstehen soll, ab sofort nicht mehr nötig ist. Die Vorteile beider Welten wurden vereint und es wird lediglich eine Anwendung entwickelt, die mit unterschiedlichen Kompilieroptionen sowohl im Internet als auch lokal auf Desktop-Computern, PDA's etc. lauffähig ist. Jede Anwendung besteht dabei aus einzelnen Seiten und ist von einem `Application`-Objekt abgeleitet, das u.a. den Einstiegspunkt der Anwendung definiert, diese steuert und Ereignisse aufruft (vgl. [long3]). Die Seiten enthalten die Beschreibung der Benutzungsschnittstelle in Form von XAML-Elementen und -Attributen sowie prozeduralem Code innerhalb der Seite (*inline*) bzw. einer Referenz auf den zugehörigen Programmcode in einer Assembly (*code-behind*).

Jeder XAML-Tag entspricht einer .NET Framework-Klasse und stellt Methoden, Eigenschaften und Ereignisse zur Verfügung. Zur Laufzeit wird das Steuerelement hinter jedem Tag instantiiert und erhält einen Bildschirmbereich, in dem die Ausgabe gerendert wird (vgl. [long2]). Ebenso wie XML-Dokumente müssen auch XAML-Seiten wohlgeformt sein und folgen daher den gleichen Regeln (vgl. [long4]). Eine einfache XAML-Seite und der dazugehörige C#-Code sind in Abbildung 7 dargestellt. Die Instantiierung der XAML-Elemente durch einen Parser sowie die Wertzuweisung und Anbindung der hervorzurufenden Ereignisse funktioniert folgendermaßen (vgl. [xaml]):

> Der Namensraum der zugehörigen Assembly wird auf den XML-Namensraum gemappt (siehe 1 und 2 in Abb. 7). Mit Hilfe von XML-Präfixen wird ermittelt, in welchem Namensraum sich die zu instantiierende Klasse befindet (siehe 3). Eigenschaften eines erzeugten Objektes werden durch Attribute gesetzt (siehe 4). Da jedes Attribut nur Zeichenketten als Werte annehmen kann, werden Type- Converter benutzt, um die Zeichenketten auf andere Typen zu mappen (siehe 5). Die Eventanbindung erfolgt ebenso über Attribute (siehe 6).

Alle Rechte vorbehalten

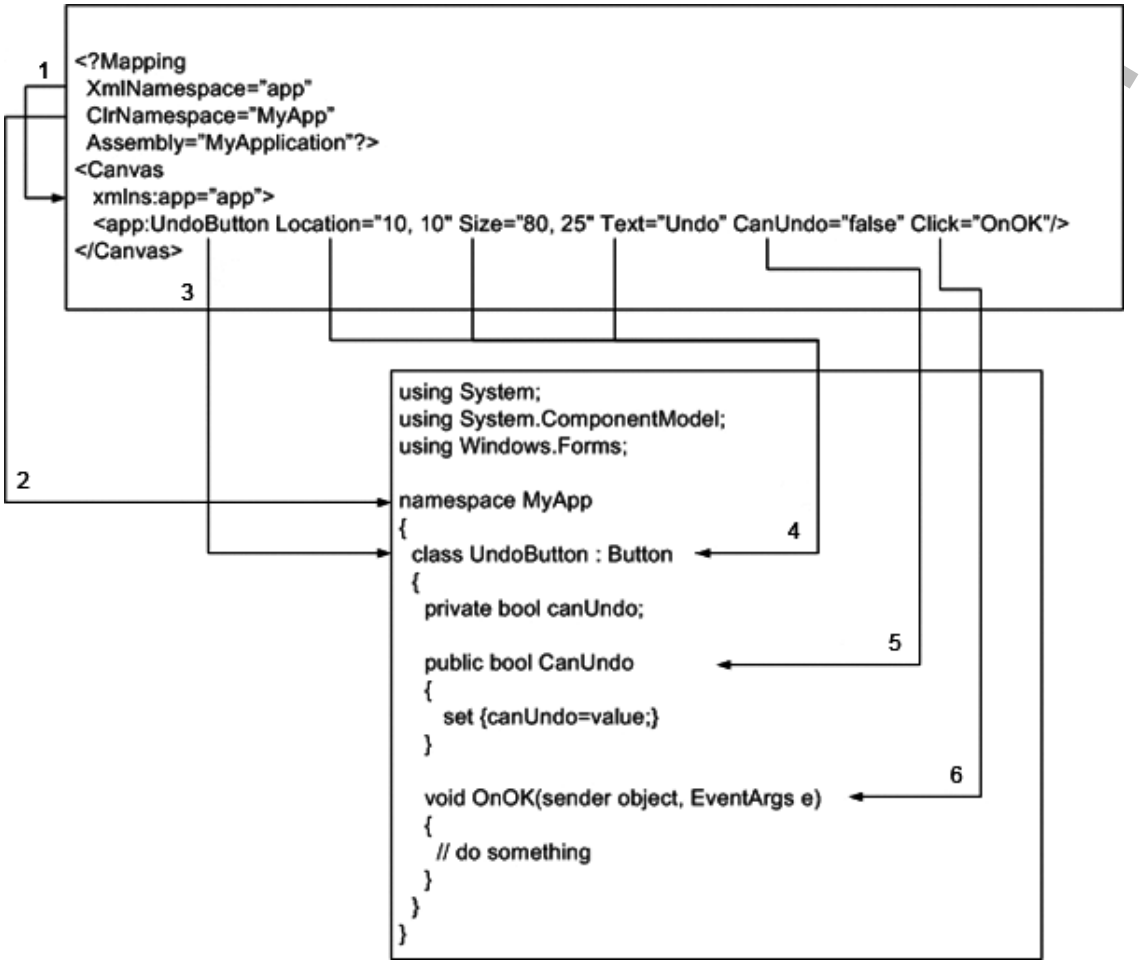

Abbildung 7: Beispiel für ein einfaches XAML-Dokument [xaml]

In XAML stehen vier Kategorien von Elementen zur Verfügung – Panel-Elemente, Interaktionselemente, dokumentbezogene Elemente und grafische Formen. Ein Fenster wird in XAML durch ein Panel-Element, z.B. `Canvas`, definiert statt wie bisher durch ein `Form`. Eine weitere Neuerung ist die Verwendung so genannter *Styles*. Diese sind Sammlungen von Stilattributen für ein Steuerelement, die der Compiler automatisch jedem Steuerelement zuweist, das den jeweiligen Stil verwendet (siehe Abbildung 8) (vgl. [long2]).

```
 1| <!-- Stildefinition -->
 2| <Style def:Name="BtnStyle">
 3|  <Button Background="Blue" FontSize="12"/>
 4| </Style>
 5|
 6| <!-- normale Buttondefinition -->
 7| <Button Background="Red">Normal</Button>
 8|
 9| <!-- Buttondefinition mit Stilzuweisung -->
10| <Button Style="{BtnStyle}">Styled</Button>
```

Abbildung 8: Beispiel für die Verwendung von Styles in XAML (vgl. [long4])

Reine XAML-Seiten, die keinerlei Programmcode enthalten, können direkt vom Browser dargestellt werden. Dies ist sehr von Vorteil, wenn rein informative, statische Benutzungsschnittstellen erzeugt werden sollen. Wird Code verwendet, entweder inline oder code-behind, so muss die Anwendung zunächst kompiliert werden, bevor diese im Browser oder als eigenständige Anwendung ausgeführt werden kann (vgl. [long3]). Weiterhin ermöglicht XAML eine saubere Trennung zwischen der Darstellung und der Benutzungsschnittstelle selbst, sofern kein inline-Code verwendet wird (vgl. [xaml]). Es können einfach und schnell Prototypen für grafische Benutzungsschnittstellen für jede Art von Programm erstellt und sogar zur Laufzeit des Programms ausgetauscht werden (vgl. [long2]). Einzelne Komponenten einer Benutzungsschnittstelle können ebenso einfach wiederverwendet werden, da lediglich das Markup kopiert und an gewünschter Stelle eingefügt werden muss. Mit Hilfe von Styles können zudem existierende Benutzungsschnittstellen schnell und einfach benutzerdefiniert modifiziert werden (vgl. [xaml]).

XAML ist aufgrund der zuvor genannten Vorteile eine Möglichkeit, grafische Benutzungsschnittstellen von Anwendungen flexibler zu gestalten und diese austauschbar zu machen. Eine Implementierung, die Microsoft's XAML-Technologie bereits verwendet und nachfolgend vorgestellt werden soll, ist MyXaml.

3.4.3. MyXaml

MyXaml[16] ist ein generisches Werkzeug zum Instantiieren von Klassen durch Textauszeichnung, stellt jedoch im Gegensatz zu XAML keine eigenständige Markup-Sprache sondern vielmehr einen Parser dar (vgl. [myxaml1] und [myxaml8]). MyXaml ist eine OpenSource-Implementierung und kann neben der dynamischen Erzeugung von Benutzungsschnittstellen zur Laufzeit u.a. auch zur Herstellung von Datenbankverbindungen oder mit Hilfe des zugehörigen Werkzeuges MxLint zur Überprüfung eines XML-Dokumentes auf Wohlgeformtheit verwendet werden. Weiterhin umfasst MyXaml Werkzeuge, wie ein Plug-In für Visual Studio .NET 2003 zur Umwandlung eines im Designer erstellten Formulares in XAML-ähnliches Markup, nachfolgend MyXaml-Markup genannt, einen einfachen Internetbrowser und kleine Spiele (vgl. [myxaml2]). Das „MyXaml-Paket“ gibt es bereits für das .NET Framework 1.1 und 2.0. Diese Arbeit konzentriert sich jedoch auf die Adaption des eigentlichen Parsers, die für .NET CF 1.0 verfügbar ist. Da .NET CF in seinem Funktionsumfang reduziert ist, fehlen auch bei dieser Implementierung einige Funktionen

16 Siehe http://www.myxaml.com,
Marc Clifton, siehe http://www.marcclifton.com/Resume/tabid/55/Default.aspx

Alle Rechte vorbehalten

bzw. wurden auf andere Weise umgesetzt[17], die grundlegende Funktionsweise des Parsers ist jedoch gleich.

Anwendungen werden in MyXaml mit Hilfe einer deklarativen und einer oder mehrerer imperativen, reflektionsfähigen[18] Sprachen entwickelt (vgl. [myxaml3] und [doc]). Erstere bezeichnet eine Markup-Sprache, wie z.B. XAML, zur Beschreibung der Benutzungsschnittstelle, letztere eine .NET-kompatible Programmiersprache wie C# zum Programmieren der Applikationslogik. Zusätzlich zum Programmcode der Anwendung kann Code inline oder code-behind vorliegen. Neben dem Namensraum System.Windows.Forms des .NET Frameworks unterstützt MyXaml im Gegensatz zu XAML die Verwendung von Komponenten von Drittanbietern, zu denen auch selbst erstellte Steuerelemente zählen. Stylesheets sind ebenso im Funktionsumfang von MyXaml enthalten (vgl. [myxaml4]).

Jeder Tag in einem MyXaml-Dokument[19] repräsentiert immer eine Klasse im angegebenen Namensraum (vgl. [myxaml5]). Es können allerdings ausschließlich Klassen instantiiert werden, die über einen parameterlosen Konstruktor verfügen. Die Eigenschaften werden über Attribute gesetzt und müssen dazu in der entsprechenden Klasse durch öffentliche Methoden, so genannte Setter, definiert sein. Ein weiterer Vorteil gegenüber XAML ist der *Include*-Tag, mit dessen Hilfe es möglich ist, mehrfach benötigte Komponenten einmalig zu beschreiben und beliebig oft wieder zu verwenden. Der Aufbau eines Dokumentes wird nachfolgend erläutert.

```
 1| <?xml version="1.0" encoding="utf-8"?>
 2| <MyXaml xml:lang="en-US"
 3|         xmlns:def="Definition"
 4|         xmlns="System.Windows.Forms">
 5|  <Form def:Name="MainForm" Load="MainForm_Load">
 6|  <Style def:Name="btn1Style">
 7|   <StyleProperties>
 8|    <PropertyStyle BackColor="Red" ForeColor="Blue"/>
 9|   </StyleProperties>
10|  </Style>
11|   <Controls>
12|    <Button def:Name="btn1" MxStyle="{btn1Style}"
13|            Location="10,10" Size="50,20" Text="Click"/>
14|    <Include Src="buttons.mxc" ComponentName="btn" Postfix="2"/>
15|   </Controls>
16|  </Form>
17| </MyXaml>
```

Abbildung 9: Beispiel für ein MyXaml-Dokument

17 Siehe http://www.codeproject.com/netcf/UsingMyXamlCF.asp

18 Mit Hilfe von Reflektion können Informationen über geladene Assemblies und die darin definierten Typen, z.B. Klassen ermittelt werden. Instanzen eines Typs können zur Laufzeit erstellt, aufgerufen und manipuliert werden (vgl. [refl]).

19 Sofern nachfolgend nicht explizit eine andere Quelle genannt wird, erfolgt die Erläuterung von Funktionsweise und Dokumentaufbau auf Basis der MyXaml-Dokumentation (siehe [doc]).

Arbeit vorbehalten

Ein MyXaml-Dokument beginnt mit der XML-Deklaration (Abb. 9 Zeile 1), gefolgt vom Element `MyXaml` (Abb. 9 Zeile 2), das den Wurzelknoten bildet und Sprache sowie Namespaces festlegt. Sind Namespace und Assemblyname gleich, so ist nur die Angabe des Namespace erforderlich, anderenfalls ist zum Mappen die Angabe beider Namen erforderlich. Werden mehrere Namespaces verwendet werden, so lassen sich diese durch Präfixe unterscheiden: `xmlns[:Präfix]="Namespace-Name, Assembly-Name"`.

Um die Anwendung in einem Fenster starten zu können, folgt im Allgemeinen die Beschreibung eines Formulares (Abb. 9 Zeile 5). Jedem Formular muss ein Name zugewiesen werden. Das `def`-Präfix bewirkt, dass dieser Name zur Definitionsliste des Parsers hinzugefügt wird und somit das Formular im zugehörigen Code über die Methode `GetReference("Objektname")` identifiziert werden kann. Existiert ein Formular bereits und wurde der Definitionsliste hinzugefügt, so kann dieses im MyXaml-Dokument per `ref:Name` anhand des jeweiligen Namens identifiziert und dem betreffenden Formular z.B. ein Stil zugewiesen werden. Wie in XAML werden Events in Form von Zeichenketten als Attribute angebunden. Wird mit Styles gearbeitet können diese nachfolgend definiert (Abb. 9 Zeile 6) oder ein externes Stylesheet über den `Include`-Tag eingebunden werden.

Im Anschluss daran werden alle zum Formular gehörigen Steuerelemente beschrieben und mit Hilfe der umschließenden `Controls`-Tags dem Formular hinzugefügt. Steuerelemente können ebenso wie Formulare definiert bzw. referenziert, Eigenschaften über Attribute gesetzt und die Ereignisse festgelegt werden. Die Zuweisung von Styles erfolgt mit Hilfe des `MxStyle`-Attributes, wobei als Wert der Name des Stils in geschweiften Klammern übergeben wird (Abb. 9 Zeile 12). In Zeile 14 wird eine externe Buttondefinition (Abb. 10) eingebunden. Die Zeichenkette `#postfix#` wird dabei mit dem Wert des `Postfix`-Attributes ersetzt, um die jeweiligen Komponenten unterscheiden zu können.

```
 1| <?xml version="1.0" encoding="utf-8"?>
 2| <MyXaml xml:lang="en-US"
 3|         xmlns:def="Definition"
 4|         xmlns="System.Windows.Forms">
 5|  <MxComponent Name="btn#postfix#">
 6|   <Button def:Name="compBtn"
 7|           Size="40,40"
 8|           Location="100,60"
 9|           Font="Tahoma, 9pt, Bold"
10|           Text="Cancel"
11|           BackColor="White"
12|           ForeColor="Green"/>
13|  </MxComponent>
14| </MyXaml>
```

Abbildung 10: Beispiel für eine MyXaml-Komponente

Alle Rechte vorbehalten

In der Beschreibung einer solchen Komponente muss zusätzlich ein `MxComponent`-Element definiert werden (Abb. 10 Zeile 5). Anhand des Namens der Komponente kann diese angesprochen, eingebunden und im aufrufenden Dokument um zusätzliche Eigenschaften erweitert werden. Eine Datei kann beliebig viele solcher Komponentenbeschreibungen enthalten. In Verbindung mit folgendem C#-Code kann das Markup verwendet werden:

```
 1|  using System.Windows.Forms;
 2|  using MyXaml;
 3|
 4|  namespace myxamltest {
 5|    public class LoadMyXamlFile {
 6|      public LoadMyXamlFile(){
 7|        MyXaml.Parser parser = new Parser();
 8|        Application.Run((Form)parser.LoadForm("testfile.myxaml", "MainForm", this, null));
 9|      }
10|
11|      static void Main() {
12|        LoadMyXamlFile loader = new LoadMyXamlFile();
13|      }
14|
15|      public void MainForm_Load(object sender, System.EventArgs e) {
16|        MessageBox.Show("Anwendung gestartet");
17|      }
18|    }
19|  }
```

Abbildung 11: Beispiel für den zu den Abbildungen 9 und 10 gehörigen C#-Code

Die `Main`-Methode stellt den Einstiegspunkt der Anwendung dar und instantiiert die Klasse `LoadMyXamlFile` (Abb. 11 Zeile 12). Im Konstruktor der Klasse wird ein neues `Parser`-Objekt (Zeile 7) erzeugt und die Anwendung gestartet (Zeile 8). Übergeben werden dazu das MyXaml-Dokument (siehe Abb. 9), der Name des Formulares, das Zielobjekt sowie das `Parent`-Objekt des Formulares. Zur Laufzeit werden die Klassen zu den Tags im MyXaml-Dokument instantiiert, Eigenschaften initialisiert und die Eventbehandlung angebunden. Dies geschieht mit Hilfe der Methode `LoadForm`. Zur Instantiierung der Klassen wird Reflektion verwendet. Dies bedeutet, dass kein unterliegendes Objektmodell der Klassen erforderlich ist, sondern zur Laufzeit Informationen aus den zu instantiierenden Klassen über verfügbare Eigenschaften und Events eingeholt werden und jene, basierend auf den ermittelten Informationen, initialisiert bzw. angebunden werden können (vgl. [myxaml3]). Die auf diese Weise erzeugten Objekte werden als Knoten eines Objektbaumes[20] im Speicher abgebildet. In Abbildung 9 wird das `Load`-Ereignis für das Formular definiert (Abb. 9 Zeile 5). Die Anwendung liefert die entsprechende Methode (Abb. 11 Zeile 15). Beim Start wird für jedes Steuerelement das `Load`-Event aufgerufen, sofern es definiert ist (vgl. [myxaml7]). Im obigen Beispiel wird dem Benutzer eine kurze Mitteilung gezeigt (Abb. 11 Zeile 16).

20 MyXaml verwendet den System.Xml-Namespace des .NET-Frameworks. Dieser baut u.a. auf den Standards XML 1.0 von 1998 sowie DOM Level 1 und 2 Core auf (vgl. [sysxml]).

Alle Rechte

Werden Anwendungen wie die zuvor beschriebene mit der MyXaml-Implementierung für das .NET Compact Framework erstellt, so sind diese unter Verwendung der entsprechenden Parser-Version und anderen Kompilieroptionen auch auf Desktop-Computern ohne Änderungen lauffähig. Die grafische Benutzungsschnittstelle kann einfach und schnell an die neue Bildschirmgröße angepasst werden (vgl. [myxaml6]). MyXaml stellt eine Alternative zu XAML dar, da es die Verwendung dieser Technologie bereits ermöglicht und sogar eine .NET CF-Implementierung bereitstellt. Die Unterstützung beliebiger Namespaces von Drittanbietern ermöglicht die Instantiierung von Klassen aus dem MSAvalon-Namespace des kommenden Windows „Longhorn“ und macht MyXaml-Anwendungen somit kompatibel zu Microsoft's XAML (vgl. [myxaml5]).

Auf Basis der in den Grundlagen vorgestellten Technologien folgen im anschließenden Kapitel eine genaue Analyse des prototypischen ESG-Client sowie die daraus resultierende Zieldefinition für die vorliegende Arbeit.

Alle Rechte vorbehalten

4. Analyse

In diesem Kapitel wird zunächst die Aufgabe beschrieben, deren Lösung Gegenstand der vorliegenden Arbeit sein soll. Anhand einer Definition der Zielgruppe für die modifizierte Anwendung, der Auswahl eines Lösungsansatzes sowie einer Analyse der bestehenden Anwendung soll ermittelt werden, welche Anforderungen an die neugestaltete Anwendung zu stellen sind und im Anschluss daran das konkrete Ziel dieser Diplomarbeit definiert werden. Die Ergebnisse bilden die verbindliche Grundlage für die anschließende Modifikation der Anwendung.

4.1. Aufgabenstellung

Wie bereits im Kapitel Hintergrund (siehe 2.1.) erwähnt, handelt es sich bei dem erstellten ESG-Client um eine prototypische Entwicklung im Rahmen eines Studienprojektes, bei der das Hauptaugenmerk vor allem auf der Erstellung eines Konzeptes und der Implementierung der grundlegenden Funktionalität lag. In der Entwicklung des Designs der grafischen Benutzungsschnittstelle hatte das Projektteam der FHTW freie Hand und sollte lediglich einen ersten Entwurf liefern. Ein konkretes Design sollte daraufhin in Abstimmung mit der Marketingabteilung der TSI vom nachfolgenden Projektteam vorgelegt werden. Da sich das Design bis zur angedachten Veröffentlichung des ESG-Client mit großer Wahrscheinlichkeit noch mehrere Male ändern wird, ist eine weitere Vertiefung des angewandten MVC-Konzeptes erforderlich, um eine problemlose Anpassung an das jeweilige neue Design zu garantieren. Besonders im Hinblick auf den Verkauf der ESG-Client-Implementierung an verschiedene Firmen muss das Design individuell anpassbar sein.

Im Rahmen dieser Arbeit soll es daher ermöglicht werden, das vorliegende Plug-In-Konzept zu adaptieren und auf die GUI anzuwenden, um diese ebenso austauschbar zu machen. Das Design der Benutzungsschnittstelle soll variierbar sein, ohne die Funktionsweise der Anwendung zu beeinflussen und dabei den Programmieraufwand möglichst gering halten. Wünschenswert wäre eine Lösung, die es ermöglicht, dass die erstellte GUI sowohl auf einem Pocket PC als auch auf einem Desktop- oder Tablet-PC benutzt werden kann, wobei eine Unterstützung der Plattform Pocket PC zwingend ist. Dieser Forderung nach individueller Anpassbarkeit soll in der vorliegenden Arbeit nachgegangen werden, indem der bestehende Prototyp derart umgestaltet wird, dass dieser die Verwendung von Skins unterstützt. Dazu soll zunächst die Zielgruppe für den modifizierten ESG-Client definiert werden.

Alle Rechte vorbehalten

4.2. Definition der Zielgruppe

Eine Modifikation des bestehenden ESG-Client im Hinblick auf eine individuelle Anpassbarkeit der grafischen Benutzungsschnittstelle wäre für die folgenden drei Gruppen von Interesse:

T-Systems International GmbH Media & Broadcast

Für die TSI wäre eine Modifikation des ESG-Client von Interesse, damit dieser an verschiedene Firmen verkauft werden kann. Gegenwärtig ist das Design auf das Corporate Design der TSI abgestimmt. Es ist daher notwendig, dass die Gestaltung der Anwendung an das Corporate Design der Käuferfirmen angepasst werden kann.

Die kaufenden Firmen

Für eine Käuferfirma wäre ein „skinningfähiger" ESG-Client insofern von Interesse, dass im Falle einer Änderung des Corporate Designs der Firma oder zu besonderen Anlässen das Design entsprechend der gegebenen Situation angepasst werden kann.

Die Endbenutzer

Als Zielgruppe kommen ebenso die Endbenutzer des ESG-Client in Frage. Wie auch in anderen Anwendungen könnte dem Benutzer die Möglichkeit geboten werden, die Gestaltung der GUI entsprechend der eigenen Vorlieben selbst zu verändern.

Die Anpassung der grafischen Benutzungsschnittstelle sollte in jedem Fall sowohl von einem erfahrenen Programmierer als auch von einem Screendesigner oder einer sonstigen Person durchgeführt werden können, die zwar grundlegende Kenntnisse im Umgang mit Computern, jedoch kaum Programmiererfahrung besitzt. Da der ESG-Client in erster Linie für die TSI entwickelt wurde, soll jene im Rahmen dieser Arbeit die Zielgruppe darstellen. Die Modifikation soll so umgesetzt werden, dass mit Hilfe einer ersten rudimentären Implementierung ein einfaches Branding der Anwendung von einem TSI-Mitarbeiter mit minimalen Programmierkenntnissen realisiert werden kann. Welcher Lösungsansatz für eine solche Modifikation des ESG-Client verwendet werden kann, wird im Folgenden ermittelt.

4.3. Auswahl eines Lösungsansatzes

Das Ziel dieses Kapitels ist es, einen Lösungsansatz zur Umsetzung der Neugestaltung des ESG-Client zu finden. Dazu soll zunächst das Thema Skinning genauer betrachtet werden, um erste Kriterien zur Auswahl eines Lösungsansatzes zu ermitteln.

Alle Rechte vorbehalten

Ein Skin umfasst verschiedene Stile, die den einzelnen Elementen einer grafischen Benutzungsschnittstelle zugeordnet werden können. Liegt die Definition dieser Stile getrennt von der Applikationslogik eines Programms vor, so kann die GUI einfach und schnell an unterschiedliche Gegebenheiten angepasst werden, indem die einzelnen Stildefinitionen verändert werden. Die unterschiedlichen Zusammenstellungen einzelner Stile können beispielsweise als Stylesheets gespeichert werden, wie sie u.a. aus dem Internet für Webseiten in Form von Cascading Stylesheets (CSS) bekannt sind.

Im Falle einer vollständigen Trennung von GUI und zugehöriger Applikationslogik wäre es möglich, bereits während der Analysephase von Softwareentwicklungsprojekten erste Prototypen der GUI zu erstellen, die nach erfolgreicher Abnahme durch den Kunden von den Programmierern mit den entsprechenden Funktionen versehen werden können. Die Prototypen können ebenso parallel zur Implementierung der Logik verändert oder neu erstellt werden, ohne die Programmierer in ihrer Arbeit zu behindern. Es könnten daher auf diese Art und Weise verschiedene grafische Benutzungsschnittstellen für ein Programm für unterschiedliche Plattformen und verschiedene Designvorstellungen der Kunden entwickelt werden, die einfach und schnell ausgetauscht bzw. modifiziert werden können.

Diese Funktionalität bietet sich für die Modifikation des ESG-Client im Hinblick auf die Anpassung an die verschiedenen Corporate Designs der kaufenden Firmen sowie die geforderte Unterstützung unterschiedlicher Plattformen an. Nachfolgend können daher bereits folgende Kriterien zur Suche einer Lösung aufgestellt werden:

- Trennung von GUI und Applikationslogik
- Verwendung von Stylesheets
- Plattformunabhängigkeit, wobei die Unterstützung der Plattform Pocket PC zwingend ist

Weitere Kriterien lassen sich aus dem Hintergrund des DVB-H-Projektes (siehe 2.1.) ableiten:

- Da der ESG-Client vollständig in C# implementiert ist, durch die Anwendung des Plug-In-Konzeptes die einzelnen Komponenten jedoch auch in anderen Programmiersprachen entwickelt sein können, soll der zur Modifikation verwendete Lösungsansatz nicht an eine bestimmte Programmiersprache gebunden sein, in jedem Fall aber C# unterstützen.
- Damit zukünftig das Branding von den als Zielgruppe festgelegten TSI-Mitarbeitern mit grundlegenden bis keinen Programmierkenntnissen durchgeführt werden kann,

Alle Rechte vorbehalten

soll die ausgewählte Lösung einfach erlernbar und leicht verständlich sein.

- Um die Lauffähigkeit der Anwendung auf diversen Plattformen zu ermöglichen, soll, wenn möglich, ein offener Standard gewählt werden. Diese Standards sind zumeist frei verwendbar und somit gut unterstützt, da sie einer großen Entwicklergemeinde zur Verfügung stehen. Es kann sich ebenso um eine OpenSource-Lösung handeln, da der bestehende ESG-Client bereits OpenSource-Komponenten verwendet.

Der einzige Lösungsansatz, der die genannten Kriterien erfüllt, ist die bereits weit verbreitete Idee der Beschreibung grafischer Benutzungsschnittstellen mittels einer XML-basierten Markup-Sprache. Dieses Konzept ist u.a. bekannt aus Anwendungen wie dem MP3-Player WinAmp oder Applikationen des Mozilla Projektes, z.B. dem Internetbrowser Firefox bzw. dem E-Mail-Programm Thunderbird. Benutzer dieser Softwareprodukte erhalten dabei die Möglichkeit, eigene Stylesheets in Form von XML-basiertem Markup zu verfassen oder vorgefertigte Dokumente aus dem Internet herunter zu laden.

Eine XML-basierte Lösung bietet sich v.a. deshalb für die Modifikation des ESG-Client an, weil zusätzlich zur Erfüllung der genannten Kriterien weitere Vorteile hinsichtlich einer Strukturierung der Daten mit Hilfe von beliebig definierbaren textuellen Tags und damit einhergehend eine Erweiterbarkeit der ausgewählten Markup-Sprache bestehen.

Es gibt bereits eine Vielzahl dieser XML-basierten Lösungen zur Beschreibung von GUI's, wie z.B. XUL und XAML, wobei die XAML-Technologie aus dem Hause Microsoft die attraktivste Lösung zur Modifikation des ESG-Client darstellt, da diese die Namensräume des .NET Frameworks unterstützt und auf Basis von Markup die zu den Tags gehörenden Klassen instantiiert. Windows „Longhorn“ und XAML sind jedoch noch nicht veröffentlicht und können daher im Rahmen dieser Arbeit nicht verwendet werden. Anderen bekannte XML-basierte Lösungen können ebenfalls nicht verwendet werden, da diese nicht die Entwicklung von Software für .NET CF unterstützen, und die erstellte Anwendung demzufolge nicht auf einem Pocket PC lauffähig wäre.

Es gibt allerdings zwei neuartige Implementierungen, die es zum einen ermöglichen, die XAML-Technologie bereits zu verwenden, und zum anderen die Entwicklung für .NET CF unterstützen. Diese sollen im Folgenden kurz vorgestellt werden:

Das Softwareprodukt *Xamlon*[21] ermöglicht die Beschreibung grafischer Benutzungs-schnittstellen mittels XAML-Markup. Es unterstützt die bisherigen Windows Forms-

21 Siehe http://www.xamlon.com

Alle Rechte vorbehalten

Komponenten sowie die neuen Avalon-Komponenten von Windows „Longhorn“ und ermöglicht die Veröffentlichung der erstellten GUI als Flash-Anwendung. Der Schwerpunkt ist bei Xamlon auf die Verwendung von Vektorgrafiken auf dem PDA gesetzt. Xamlon unterstützt ausschließlich die Entwicklung mit Visual Studio .NET 2003. Bei der Installation des Xamlon-Paketes auf dem PC wird ein zusätzliches Plug-In installiert mit dem GUI's „zusammengeklickt“ werden können oder in der XAML-Ansicht das Markup direkt bearbeitet werden kann. Die Software ist in drei Version für die Entwicklung von Anwendungen für Desktop-PC's, das Internet und PDA's verfügbar, muss allerdings im Allgemeinen käuflich erworben werden. Die .NET CF-Version ist ausschließlich als Beta-Version für registrierte Beta-Tester erhältlich.

Ein weiterer Lösungsansatz ist *MyXaml*. Diese Implementierung stellt ein generisches Werkzeug zur Instantiierung von Klassen aus beliebigen Namensräumen dar. Es wurde entwickelt zur Beschreibung von herkömmlichen Windows Forms-Steuerelementen, ermöglicht zusätzlich aber auch die Verwendung von Komponenten von Drittanbietern und unterstützt somit auch selbst erstellte Steuerelemente sowie die kommenden Avalon-Komponenten. Die Beschreibung der GUI erfolgt mittels XAML-ähnlichem Markup. Da es sich bei dieser Implementierung um ein OpenSource-Projekt handelt, ist diese frei verfügbar und kann im Erfahrungsaustausch mit anderen Anwendern entsprechend der erforderlichen Funktionalität weiter entwickelt werden. Die Verwendung von MyXaml ist nicht an die Entwicklungsumgebung Visual Studio .NET 2003 gebunden. MyXaml ist in zwei Versionen verfügbar, wobei zur Entwicklung von Software für .NET CF keine Installation des MyXaml-Paketes erforderlich ist, sondern lediglich eine Klassenbibliothek bei der Entwicklung in das Projekt eingebunden werden muss.

Die Modifikation des bestehenden ESG-Client soll daher mit Hilfe der MyXaml-Technologie realisiert werden. Die im Folgenden erläuterten Vorteile begründen die Auswahl dieses Lösungsansatzes:

- MyXaml unterstützt die Entwicklung von Anwendungen für .NET CF.
- Zur Entwicklung für .NET CF ist im Gegensatz zu Xamlon keine Installation der Software notwendig. Es muss lediglich eine Klassenbibliothek bei der Programmierung eingebunden werden. Da diese jedoch speziell auf .NET CF ausgerichtet ist, wird die Modifikation des ESG-Client im Rahmen dieser Arbeit voraussichtlich nur für die Plattform Pocket PC umgesetzt werden können (siehe Kapitel 2.3.). Die erstellte GUI sollte jedoch problemlos auch unter Verwendung

Alle Rechte vorbehalten

der Desktop-Version benutzbar und lediglich eine Anpassung der Größen und Positionen der beschriebenen Elemente notwendig sein.

- MyXaml ist im Gegensatz zu Xamlon an keine Entwicklungsumgebung gebunden.
- Mit MyXaml können einzelne Komponenten in verschiedenen unterliegenden Programmiersprachen entwickelt und als Assemblies veröffentlicht werden.
- Es ist im Gegensatz zu Xamlon keine Umwandlung in eine Flash-Anwendung notwendig um ein lauffähiges Programm für PDA's zu erstellen.
- MyXaml ist im Gegensatz zu Xamlon eine OpenSource-Implementierung und kann daher kostenlos verwendet und nach Belieben weiterentwickelt werden, sofern zusätzliche Funktionen zur Neugestaltung des ESG-Client notwendig sind.
- Drittanbieterkomponenten werden unterstützt. Die im ESG-Client verwendeten OpenNETCF- und selbst erstellten Steuerelemente können daher weiterhin, auch durch Beschreibung in Form von Markup problemlos benutzt werden. Mit Erscheinen von Windows „Longhorn“ sollte es daher möglich sein, den modifizierten ESG-Client mit geringstem Programmieraufwand in eine „richtige“ XAML-Applikation umzuwandeln. Lauffähig wäre die unter Verwendung von MyXaml neugestaltete Version auf einem Windows „Longhorn“-Betriebssystem jedoch auch ohne Änderungen.
- Wie auch Xamlon unterstützt MyXaml die Verwendung von Vektorgrafiken.
- MyXaml unterstützt die Wiederverwendbarkeit von Komponenten. Dasselbe Markup kann durch einfaches Kopieren und Einfügen in unterschiedlichen Anwendungen benutzt werden. Ebenso kann es auch als MxComponent in separaten Dokumenten abgelegt und über einen Include-Tag beliebig oft in andere MyXaml-Dokumente eingefügt werden.
- MyXaml stellt keine komplexe Sprache mit eigenem Objektmodell sondern einen einfachen Parser dar, der mit den Namensräumen System.Reflection und System.Xml die vom .NET Framework bereitgestellte Funktionalität verwendet.
- MyXaml unterstützt die Verwendung von Styles.

Mit Hilfe dieser XML-basierten Lösung sollte es möglich sein, dass die Gestaltung der GUI vollständig von Benutzern mit geringsten Programmierkenntnissen durchgeführt werden kann. Es wird lediglich die unterliegende Logik zum Aufruf der dynamischen Generierung der GUI

Alle Rechte vorbehalten

sowie die Ereignisbehandlung in Form von Programmcode benötigt. Im Falle eines Brandings für verschiedene Firmen könnte daher direkt durch Designer anstelle eines Photoshop-Prototypen die Erstellung eines Skins erfolgen, welcher zusammen mit der GUI-Beschreibung und dem unterliegenden Code einfach und schnell einen fertigen ESG-Client bildet.

Nachfolgend soll festgestellt werden, inwieweit eine Modifikation des bestehenden ESG-Client mit Hilfe des gewählten Lösungsansatzes möglich ist. Dazu wird analysiert, inwiefern die bestehende Anwendung bereits eine Trennung zwischen GUI und Applikationslogik aufweist bzw. welche Veränderungen zur Umsetzung dieses Konzeptes notwendig sind.

4.4. Bestehender ESG-Client

Gegenstand der folgenden Abschnitte sind die Vorstellung der Systemarchitektur des ESG-Client sowie die Auswahl und Analyse einiger zu modifizierender Komponenten im Hinblick auf die Anwendbarkeit von Skinning mit Hilfe der zuvor ausgewählten MyXaml-Technologie.

4.4.1. Systemarchitektur

Die bestehende Anwendung ist vollständig in C# implementiert. Sie setzt sich zusammen aus dem ESG-Client selbst (Hauptformular und Startbildschirm) sowie sechs Plug-Ins, die über eine Schnittstelle geladen werden. Die Plug-Ins Search, OnAir und Browse binden ein Informationen-Fenster in Form einer separaten Klassenbibliothek ein. Alle Plug-Ins verwenden das speziell für den ESG-Client geschaffene Steuerelement ImageButton. Die benötigten Informationen über verfügbare Services werden unter Verwendung der Zwischenschicht DVB-H AL ermittelt.

In Abbildung 12 wird das bestehende System in Form eines Package-Diagramms veranschaulicht. Dargestellt sind jeweils die öffentlichen Klassen eines Packages sowie die Beziehungen der Komponenten zueinander. Die Beziehungen zeigen an, ob die Komponente, auf die mittels eines Pfeiles verwiesen wird, im zeigenden Package über einen Verweis referenziert (Use-Beziehung) oder von mindestens einer Klasse des Packages über ein using-Statement eingebunden (Import-Beziehung) wird.

Alle Rechte vorbehalten

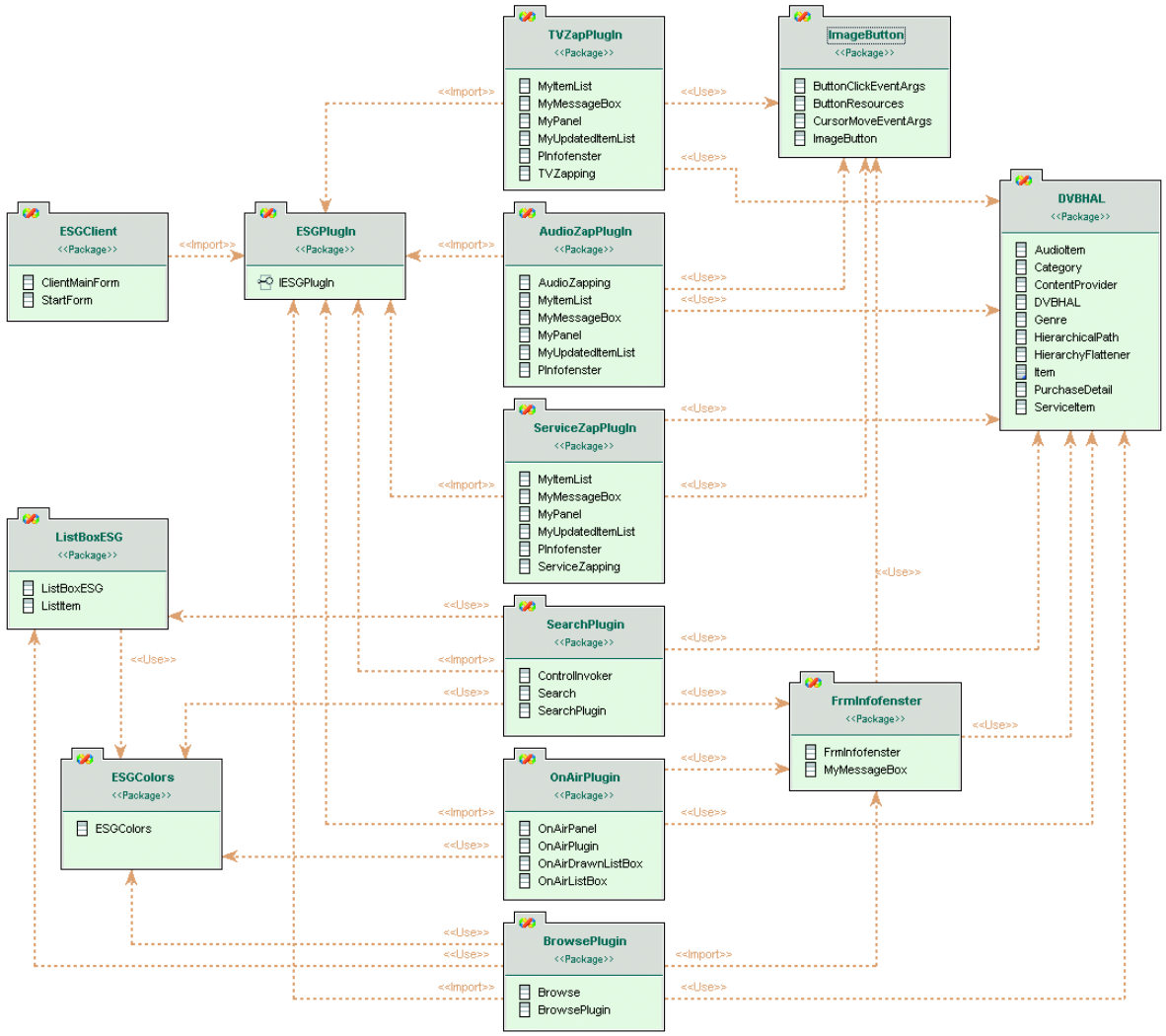

Abbildung 12: Systemarchitektur des bestehenden ESG-Client

Im Rahmen dieser Arbeit soll, aufgrund der Komplexität des Systems und des begrenzten Zeitfensters, eine mögliche Neugestaltung des ESG-Client anhand einiger weniger Komponenten demonstriert werden. Daher wird an dieser Stelle auf eine vollständige Analyse der bestehenden Anwendung verzichtet und statt dessen nachfolgend Komponenten ausgewählt und analysiert. Der ausführlich dokumentierte Programmcode des bestehenden ESG-Client ist auf der im Anhang befindlichen CD-ROM zu finden.

4.4.2. Analyse ausgewählter Komponenten

Damit als Resultat der beispielhaften Modifikation eine funktionstüchtige Anwendung vorliegt, werden der Client inklusive Startbildschirm, die Schnittstelle sowie der ImageButton und ein Plug-In ausgewählt. Die geforderte Erhaltung des Plug-In-Konzeptes soll mittels des TVZapPlugIns realisiert werden. Auf eine Analyse des DVB-H AL wird verzichtet, da die

Alle Rechte

Funktionalität dieser Schicht unverändert erhalten bleiben soll. In den folgenden Abschnitten werden die vier ausgewählten Komponenten hinsichtlich deren Modifizierbarkeit untersucht. Es soll ermittelt werden, inwiefern eine Trennung von GUI und Logik möglich bzw. schon implementiert ist. Weiterhin soll die Möglichkeit der Verwendung von Stylesheets überprüft werden. Die Komponente ImageButton wird im Hinblick auf die Kompatibilität mit MyXaml untersucht.

4.4.2.1. ESG-Client

Der eigentliche ESG-Client, der für das Laden der Plug-Ins und die Bereitstellung des DVB-H AL verantwortlich ist, besteht aus zwei Formularen, dem ClientMainForm und dem StartForm (siehe Abbildung 13). Beide Formulare beinhalten eine Titelleiste sowie einen Druckschalter zum Schließen des Formulares bzw. Beenden der Anwendung, ein FHTW-Logo sowie Druckschalter zur Repräsentation der verfügbaren Plug-Ins. Auf dem ClientMainForm bilden letztere am rechten Bildschirmrand eine Tableiste. Der linke Bereich ist ein Panel, in dem das jeweils aktive Plug-In angezeigt wird.

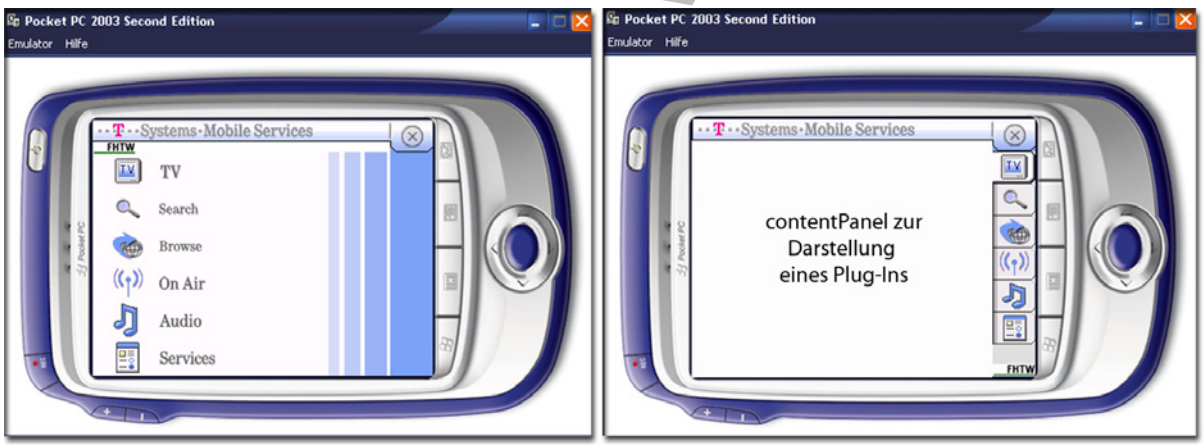

Abbildung 13: Screenshots des bestehenden StartForms (links) und ClientMainForms (rechts)

Das ClientMainForm stellt eine Methode `Main` als Einstiegspunkt zum Starten der Anwendung bereit. In dieser Methode wird mittels eines Konstruktoraufrufes ein neues `ClientMainForm`-Objekt erzeugt. Dazu werden im Konstruktor zunächst das unterliegende Betriebssystem und die Plattform überprüft sowie die Methoden `InitializeComponent` und `Init` aufgerufen, die wiederum die GUI des Formulares und die zugehörige Logik bereitstellen.

Alle Rechte vorbehalten

Die Definition der GUI erfolgt zum größten Teil in der Methode `InitializeComponent`. Darin werden ein `ResourceManager`-Objekt zur Verwaltung der verwendeten Bilder sowie die einzelnen Steuerelemente erzeugt, deren Eigenschaften initialisiert und Eventhandler zugewiesen. Weiterhin werden Eigenschaften des `ClientMainForm` selbst gesetzt, das `Load`-Event angebunden und alle Steuerelemente zum Formular hinzugefügt. Auf die Methode `InitializeComponent` könnte bei der Modifikation des ESG-Client verzichtet und an deren Stelle die darin definierten Elemente mit Hilfe von MyXaml-Markup in einem separaten Dokument beschrieben werden. Ein `ResourceManager` wird in Zukunft ebenfalls nicht mehr benötigt, da zur Durchführung des Skinnings die verwendeten Bilder nicht mehr als Ressourcen-Objekte sondern als Bilddateien in einem Ordner vorliegen müssen, um ausgetauscht werden zu können.

Die `Init`-Methode definiert mit Hilfe anderer Methoden die Steuerung der Anwendung. Es wird festgelegt, dass, sobald das Formular den Fokus erhält, dieser an das `contentPanel`, d.h. das geladene Plug-In, übergeben wird. Sämtliche Ressourcen werden geladen und den zuvor erzeugten Steuerelementen zugewiesen. In der modifizierten Version des ESG-Client könnten die Bilder direkt im MyXaml-Dokument den einzelnen Elementen durch Pfadangaben zugewiesen werden. Es folgen die Definitionen von Listen der zu ladenden Plug-Ins, die sich je nach unterliegender Plattform inhaltlich unterscheiden. Mit Hilfe der Methode `loadPlugIns` wird versucht, diese Plug-Ins zu laden. Die Tableiste am rechten Rand des Formulares repräsentiert die erfolgreich geladenen Plug-Ins in Form von Druckschaltern. Diese werden abhängig von der Plug-In-Anzahl dynamisch generiert.

Nach Fertigstellung des ClientMainForms wird der `Startscreen` erzeugt und beim Start der Anwendung im Vordergrund angezeigt. Dem Konstruktor wird dabei das `ClientMainForm`-Objekt als Parameter übergeben. Es werden ebenfalls Betriebssystem und Plattform überprüft sowie mit Hilfe einer `InitializeComponent`-Methode die Steuerelemente des Formulares definiert, Eigenschaften des Formulares und der Steuerelemente gesetzt und Eventhandler angebunden. Die Bildverwaltung wird auch im StartForm über einen Ressourcen-Manager geregelt. Über das übergebene `ClientMainForm`-Objekt kann der Startscreen auf dessen Methoden zugreifen und die geladenen Plug-Ins auslesen. In der Methode `createButtons` werden dynamisch die Druckschalter zur Repräsentation der Plug-Ins erzeugt.

Aufgrund der ermittelten Gemeinsamkeiten in Aufbau und Funktionsweise könnten für den Startscreen dieselben Änderungsvorschläge wie für das ClientMainForm gelten. Beide

Alle Rechte vorbehalten

verwenden dieselben Methoden zum Auslesen der verfügbaren Plug-Ins sowie zur Aktivierung bzw. Deaktivierung und Darstellung der vom Benutzer ausgewählten Plug-Ins. Diese Methoden sollen in ihrer grundlegenden Funktionalität erhalten bleiben. Es können jedoch Veränderungen notwendig sein, so dass zwar das Resultat gleich bleibt, dieses jedoch durch eine andere Vorgehensweise erzielt wird. Der Konstruktor des StartForms ist nicht parameterlos, sollte aber dennoch weiterhin verwendet werden können. Es ist angedacht, dass die bestehenden Klassen `ClientMainForm` und `StartForm` verwendet und darin jeweils die zugehörigen MyXaml-Dokumente zur Beschreibung der GUI eingelesen werden. Somit kann die Parameterübergabe im Konstruktor des StartForms beibehalten und ein einfaches Formular mittels MyXaml-Markup beschrieben werden. Es ist weiterhin möglich, die Anwendung auch ohne den Startscreen zu verwenden bzw. diesen auszutauschen. Steuerelemente, die bisher dynamisch erzeugt wurden, sollten auch in der modifizierten Version des ESG-Client dynamisch, jedoch in Form von Markup-Elementen generiert und in das Dokument eingefügt werden.

In der folgenden Abbildung sind die beiden Klassen mit allen Variablen und Methoden sowie deren Beziehung zueinander in einem Klassendiagramm dargestellt:

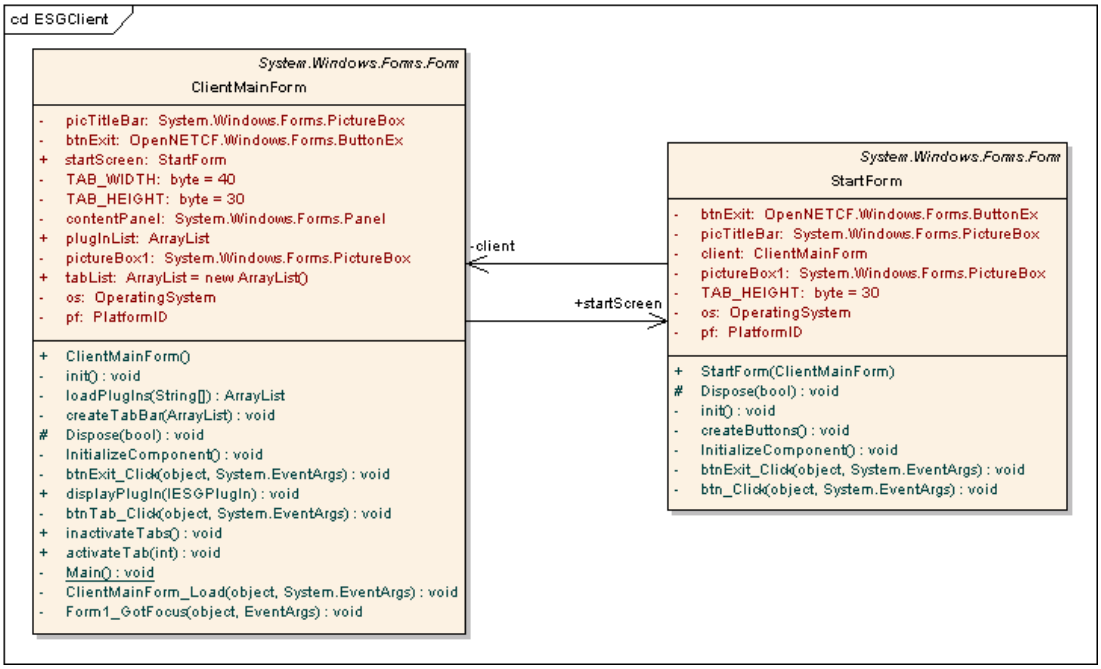

Abbildung 14: Klassendiagramm ESGClient-Package

Die Klassenvariablen sind in rot dargestellt. Öffentliche Methoden und Variablen sind mit einem „+“, private mit einem „-“ versehen. Geschützten Methoden und Variablen ist eine Raute „#“ vorangestellt. Der Einstiegspunkt der Anwendung ist unterstrichen.

Vorbehalten

Die vom ClientMainForm zu ladenden Plug-Ins werden auf ihren Typ hin überprüft. Es werden ausschließlich Plug-Ins geladen, die die Schnittstelle `IESGPlugIn` implementieren. Diese soll daher im folgenden Abschnitt genauer betrachtet werden.

4.4.2.2. IESGPlugIn

Anhand der Schnittstelle `IESGPlugIn` kann vom ClientMainForm ein Plug-In erkannt, geladen und eingebunden werden. Dazu wird die öffentliche Methode `getContentPanel` aufgerufen. Diese erzeugt ein `Panel`-Objekt und übergibt dieses an den aufrufenden Client. Eine Übersicht über alle Methoden der Schnittstelle ist in Abbildung 15 zu sehen. Ein Plug-In, welches im ESG-Client verwendet werden soll, muss alle aufgeführten Methoden implementieren.

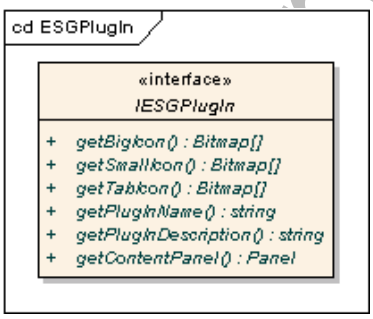

Abbildung 15: Klassendiagramm ESGPlugIn-Package

Die zur Repräsentation eines Plug-Ins auf dem Startscreen bzw. in der Tableiste benötigten Bilder werden von den Plug-Ins als Ressourcen eingebunden. Die Methoden `getBigIcon` und `getTabIcon` übergeben diese Bilder in Form von Bitmap-Arrays. Da später die Bilder, wie bereits erwähnt, zwecks Austauschbarkeit beim Branding, in einem Ordner vorliegen sollen, könnten an dieser Stelle String-Arrays zurückgegeben werden, welche die Pfade zu den jeweiligen Bildern enthalten. Die Methode `getSmallIcon` wird im bestehenden ESG-Client nicht verwendet. Der Name eines Plug-Ins wird vom Client anhand der Methode `getPlugInName` ermittelt und mit der Liste der zu ladenden Plug-Ins verglichen. Die Methode `getPlugInDescription` liefert eine Beschreibung der Funktionalität eines Plug-Ins. Soll später ein Stylesheet verwendet werden, muss dieses voraussichtlich mittels einer Methode dieser Schnittstelle an die Plug-Ins übergeben werden.

Ein Plug-In, welches die Schnittstelle `IESGPlugIn` implementiert ist das im Folgenden vorgestellte TVZapPlugIn.

Alle Rechte vorbehalten

4.4.2.3. TVZapPlugIn

Wie an der zuvor abgebildeten Systemarchitektur (siehe 4.4.1. Abbildung 12) zu erkennen ist, sind alle drei Zapping-Plug-Ins identisch aufgebaut. Sie unterscheiden sich lediglich im Aufruf einer Methode des DVB-H AL, da jeweils verschiedene Typen von Services angezeigt werden sollen. Aus diesem Grund wird im Rahmen dieser Arbeit stellvertretend das TVZapPlugIn genauer betrachtet und modifiziert. Die beiden anderen Zapping-Plug-Ins können im Anschluss daran einfach und schnell, basierend auf der Vorlage des TVZapPlugIns, umgesetzt werden.

Das Plug-In-Panel besteht aus einer Liste, welche die Anbieter aller verfügbaren Items anzeigt, sowie einem weiteren Panel, in dem Informationen zu den entsprechenden Items dargestellt werden. Der folgende Screenshot gibt einen Überblick über die Steuerelemente, die auf den einzelnen Panels angeordnet sind.

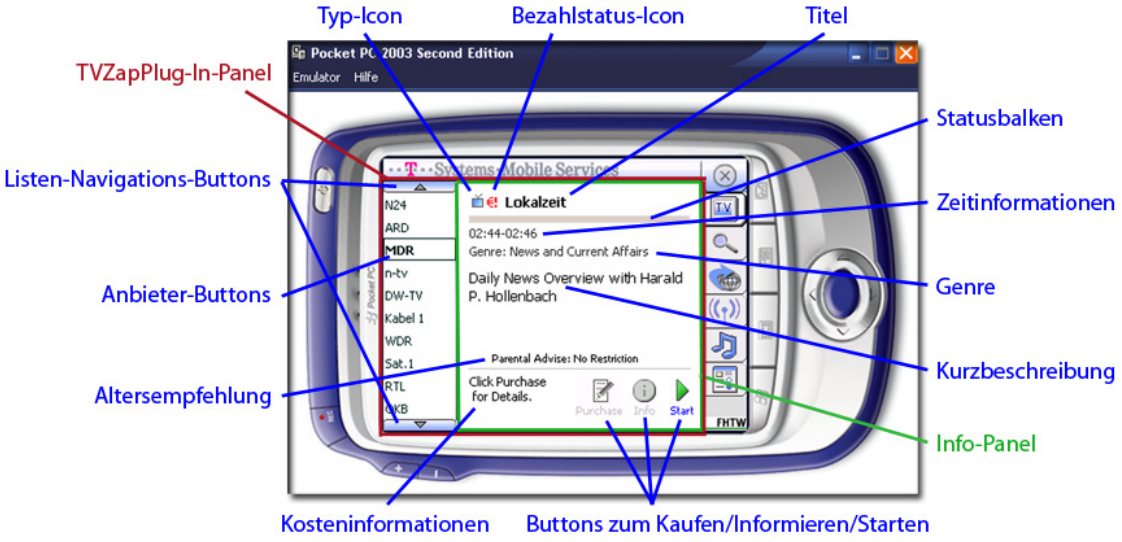

Abbildung 16: Screenshot des bestehenden TVZapPlugIns

Das TVZapPlugIn gibt eine Übersicht über alle verfügbaren TV-Services. Die dazu benötigten Daten werden in diesem Fall mit Hilfe der Methode `getCurrentVideoItems` aus dem DVB-H AL abgerufen. Diese Vorgehensweise soll unverändert bleiben. Das Plug-In ist definiert durch die in Abbildung 17 dargestellten sechs Klassen. Den Einstiegspunkt in das Plug-In bildet dabei die Methode `getContentPanel` der Schnittstelle `IESGPlugIn`. Bei deren Aufruf durch den Client wird eine Instanz der Klasse `MyPanel` erzeugt. Das entstehende Panel stellt das Plug-In dar, welches daraufhin vom ClientMainForm eingebunden und angezeigt wird.

Alle Rechte vorbehalten

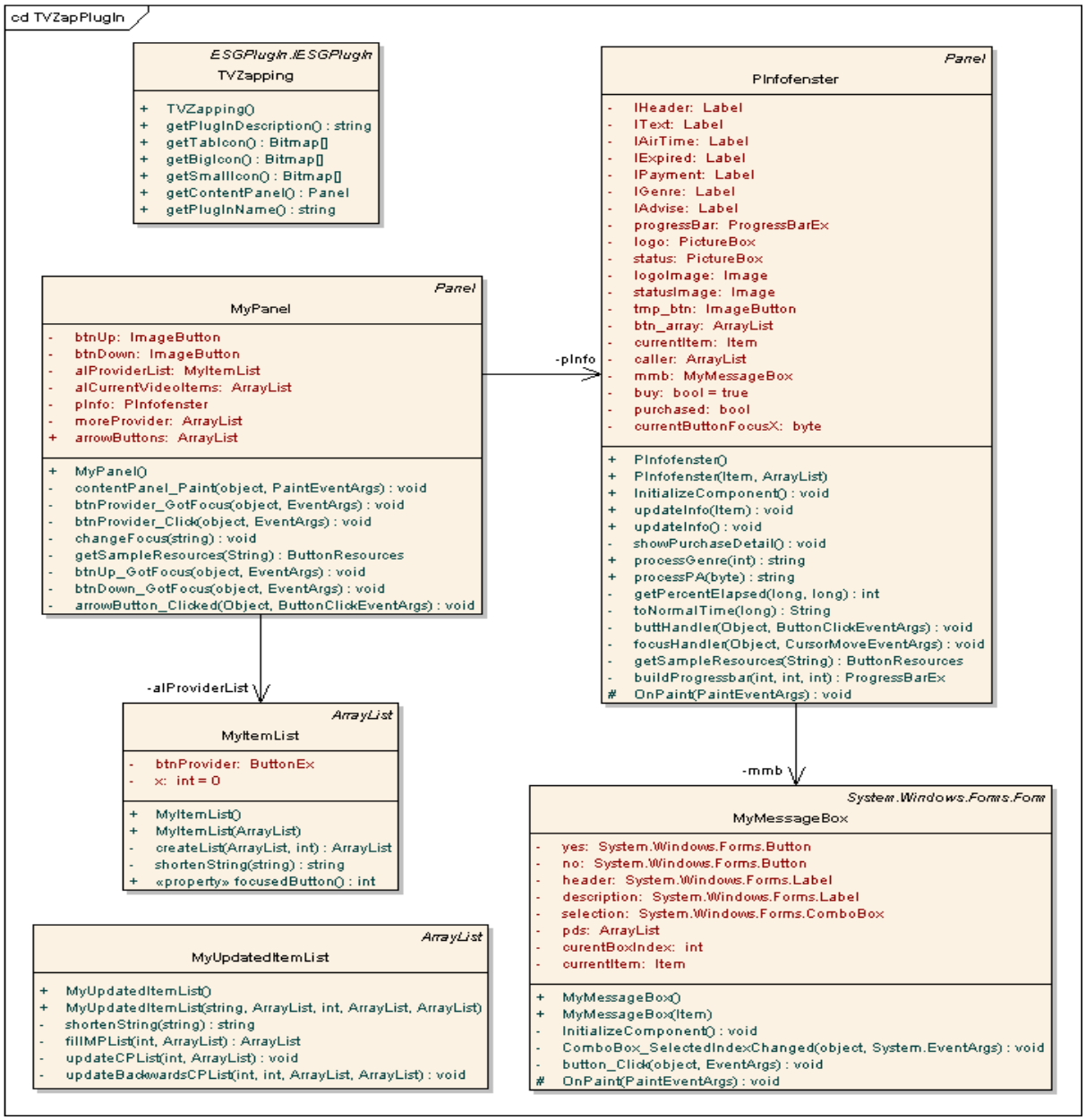

Abbildung 17: Klassendiagramm TVZapPlugIn-Package

Im Konstruktor der Klasse `MyPanel` werden zunächst die Eigenschaften des Panels gesetzt. Diese könnten in der modifizierten Version der Anwendung direkt im MyXaml-Dokument in der Beschreibung des Panels als Attribute übergeben werden. Es folgt die Anbindung des `Paint`-Events, bei dessen Aufruf das Panel gezeichnet wird. Dabei wird zuerst geprüft, ob Items vom Typ `VideoItem` vorliegen. Ist dies nicht der Fall, so wird ein Label mit entsprechendem Text erzeugt. Wenn Items verfügbar sind, so wird geprüft ob es sich um ein Update des Panels handelt und lediglich das zugehörige Informationsfenster aktualisiert. Wird das Panel zum ersten Mal erzeugt, so werden die für die Zapping-Funktionen benötigten Steuerelemente erzeugt und zum Panel hinzugefügt. Zur Darstellung der Liste werden dabei ImageButtons zur Navigation nach oben und unten erzeugt, mit Eigenschaften und

Eventhandlern versehen sowie zu einer ArrayList und zum Panel hinzugefügt. Auf die Klasse `ImageButton` wird im folgenden Abschnitt genauer eingegangen. Die vorhandenen VideoItems werden aus dem DVB-H AL ausgelesen und in der Klasse `MyItemList` die entsprechenden Druckschalter generiert sowie mit Eventhandlern versehen und ebenfalls zum Panel hinzugefügt. Abhängig von der Situation könnten bei der Neugestaltung des ESG-Client dynamisch die benötigten Steuerelemente in Form von MyXaml-Elementen zum Dokument hinzugefügt werden.

Zur Abgrenzung der Anbieterliste vom Informationen-Panel wird eine Linie an den rechten Rand der Liste gezeichnet. Mit MyXaml kann dies nicht durchgeführt werden, daher muss speziell zum Zeichnen der Linie das `Paint`-Event definiert werden.

Es folgt die Instantiierung der Klasse `PInfofenster`. Da alle drei Zapping-Plug-Ins eine identische Implementierung verwenden wird in Erwägung gezogen, dieses als separate Komponente mit eigenem Namespace umzusetzen. In der Klasse `MyPanel` werden die Größe und die Position des PInfofensters festgelegt. Diese sollten zukünftig im zugehörigen MyXaml-Dokument oder im Stylesheet gesetzt werden. An das PInfofenster werden das aktuell darzustellende VideoItem sowie die Navigationsbuttons der Liste übergeben, da die Steuerung vollständig vom PInfofenster geregelt wird. Dieses Vorgehen sollte beibehalten werden.

Die Definition der GUI erfolgt fast ausschließlich in der Methode `InitializeComponent`. Auf diese kann daher verzichtet werden und die Beschreibung der dort definierten Steuerelemente und des Panels im MyXaml-Dokument stattfinden sowie die Eigenschaften in einem Stylesheet definiert werden. Die Anzeige des PInfofensters wird in der Methode `updateInfo(Item item)` ausgeführt. Dort werden die Bilder für den Medientyp und Bezahlstatus des aktuell übergebenen Items den Steuerelementen zugewiesen. Die Bilder sollten auch in diesem Fall wieder keine Ressourcen sein, sondern durch Angabe eines Pfades aus dem entsprechenden Ordner geladen werden. Die Zuweisung der neuen Bilder könnte weiterhin in der `updateInfo`-Methode stattfinden, damit das MyXaml-Dokument nicht bei jeder Aktualisierung neu eingelesen werden muss.

Weiterhin wird bei jedem Aufruf dieser Methode der Fortschrittsbalken gelöscht und neu gezeichnet. Es wäre ausreichend, lediglich den Wert für die Größe des Balkens neu zu berechnen. Die Beschreibung des Fortschrittsbalkens könnte daher ebenfalls im MyXaml-Dokument erfolgen. Abhängig vom gegenwärtig ausgewählten Item werden ebenso die Texte aller Labels geändert sowie der Kaufen-Button ein- oder ausgeblendet. Diese Änderungen der

Alle Rechte vorbehalten

Anzeige sollten weiterhin von der `updateInfo`-Methode vorgenommen werden. Bei den angezeigten Buttons zum Kaufen, Informieren oder Starten eines Dienstes handelt es sich um ImageButtons, die mittels eines Konstruktors mit Parameterliste erzeugt werden. Diese Problematik wird im Abschnitt 4.4.2.4. diskutiert.

Wenn der Druckschalter zum Kaufen eines Services eingeblendet wird und der Benutzer diesen drückt, wird in der Methode `showPurchaseDetail` eine „MessageBox“ erzeugt (siehe Abbildung 18).

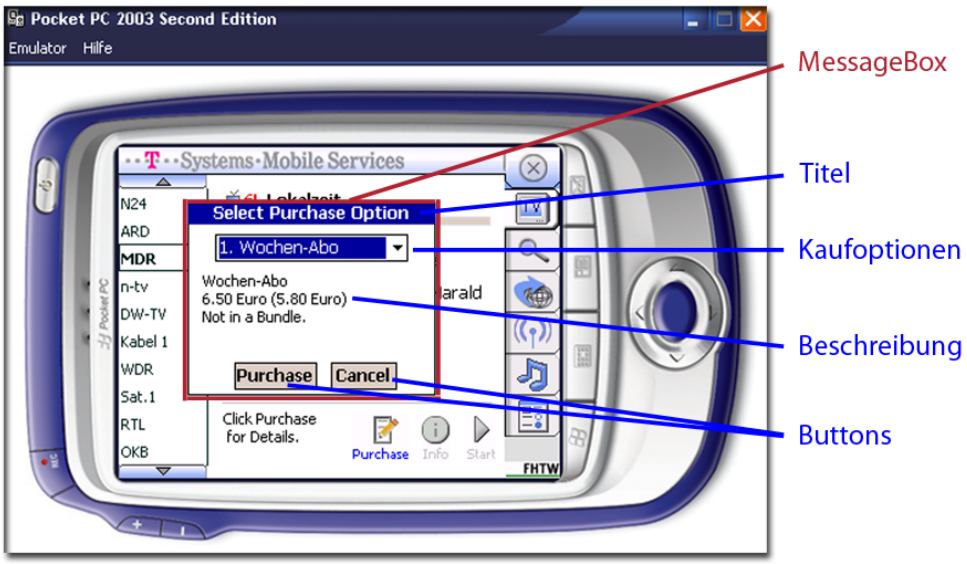

Abbildung 18: Screenshot der MessageBox des PInfofensters

Es handelt sich um eine Instanz der Klasse `MyMessageBox`, die ein Formular mit Größe und Aussehen einer MessageBox erstellt. Die Eigenschaften dieses Formulares werden ebenfalls in der Methode `showPurchaseDetail` initialisiert, sollten jedoch in der modifizierten Version des ESG-Client in einem Stylesheet festgelegt werden. Das Formular mit allen Steuerelementen sollte in einem separaten MyXaml-Dokument beschrieben werden. Momentan wird die GUI in der Methode `InitializeComponent` erzeugt. In der Methode `ComboBox_SelectedIndexChanged` werden die möglichen Kaufoptionen eines Items, die als Einträge einer ComboBox angezeigt werden, sowie die zugehörigen ausführlichen Informationen ermittelt und angezeigt. Diese Vorgehensweise sollte in dieser Form erhalten bleiben können. Sobald ein neues Item übergeben wird sollten jedoch ausschließlich die angezeigten Werte aktualisiert werden, statt wie bisher bei jedem Klick auf den Kaufen-Druckschalter eine neue MessageBox zu erzeugen.

Wie bereits in diesem Kapitel erwähnt verwenden die Klassen `MyPanel` und `PInfofenster` Instanzen der Klasse `ImageButton`. Da diese jedoch durch

Konstruktoren mit Parameterlisten erzeugt werden, muss dieser hinsichtlich einer Instantiierung durch den MyXaml-Parser modifiziert werden. Inwieweit Veränderungen notwendig sind soll im folgenden Abschnitt ermittelt werden.

4.4.2.4. ImageButton

Die Klasse `ImageButton` definiert einen Druckschalter, der vier verschiedene Zustände annehmen kann und diese dem Benutzer durch die Anzeige verschiedener Bilder mitteilt. Die Bilder können über die Get- und Set-Methoden der `ImageButton`-Klasse gesetzt oder in Form eines `ButtonResources`-Objektes übergeben werden (siehe Abbildung 19).

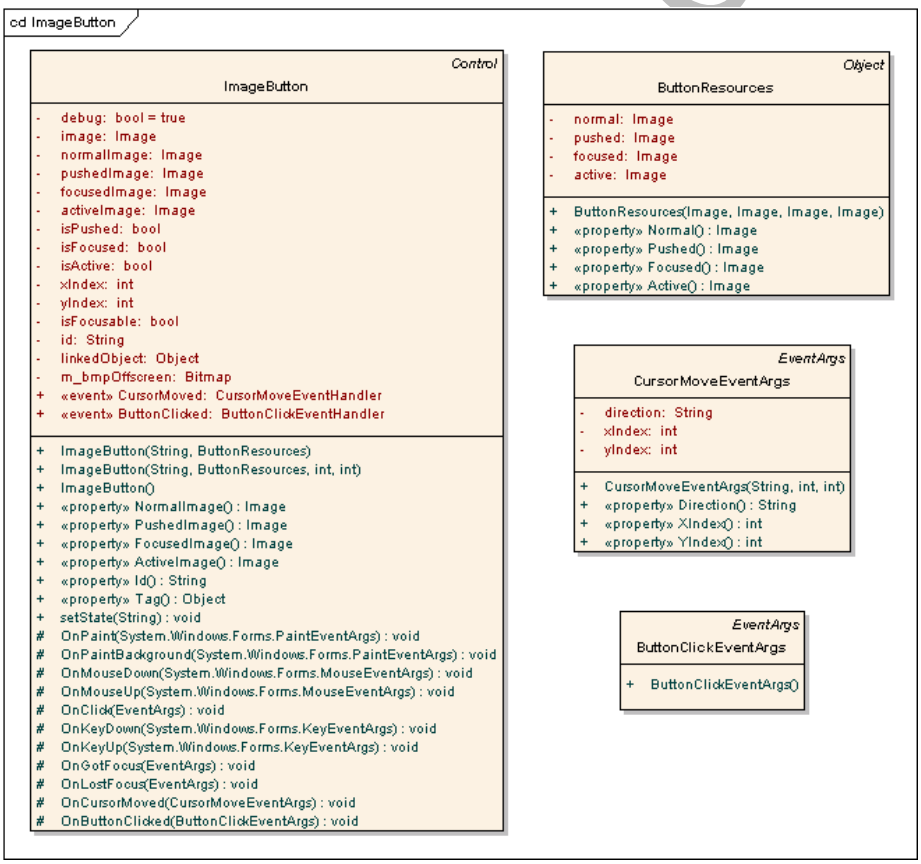

Abbildung 19: Klassendiagramm ImageButton-Package

Die Komponente ImageButton besitzt in der aktuellen Version keinen eigenen Namensraum und muss aufgrund dessen zur Instantiierung durch den MyXaml-Parser modifiziert werden. Weiterhin müssen Veränderungen am Aufbau der Klasse `ImageButton` vorgenommen werden, da gegenwärtig die benötigten Buttons in den zuvor analysierten Klassen `MyPanel` und `PInfofenster` ausschließlich mittels Konstruktoren erzeugt werden können, die eine Parameterliste benötigen. Es ist zwar ein parameterloser Konstruktor vorhanden, jedoch kann

ein ImageButton, der unter Verwendung dieses Konstruktors erzeugt wird, nicht mit der gleichen Funktionalität verwendet werden.

Es kann beispielsweise kein 2D-Fokus-Index zur Navigation zwischen verschiedenen ImageButtons manuell gesetzt werden, wie er im PInfofenster verwendet wird. Ebenso ist es bisher nicht möglich, selbst zu bestimmen ob ein ImageButton fokussierbar ist oder nicht. In der Klasse `MyPanel` wird beispielsweise ein nicht fokussierbarer Button benötigt wird, während die Klasse `PInfofenster` einen fokussierbaren ImageButton verlangt. Daher müssen in der modifizierten Version des ESG-Client ein leerer Konstruktor, eine Methode zum Setzen des Fokus-Index sowie Getter und Setter zum Abrufen bzw. Setzen der Fokussierbarkeit zur Verfügung gestellt werden. Die Klasse `ButtonResources` wird nicht mehr benötigt, da die Bilder direkt in der MyXaml-Beschreibung über die entsprechenden Setter als Pfadangaben gesetzt werden können. Es muss jedoch in der `OnPaint`-Methode geprüft werden, ob alle Zustände durch Bilder definiert wurden bzw. fehlende Bilder durch andere ersetzt werden. Diese Prüfung wird in der aktuellen Version im Konstruktor vorgenommen, wenn ein `ButtonResources`-Objekt übergeben wurde.

Im Hinblick auf den Entwurf der modifizierten Version der Anwendung sollen nachfolgend die grundlegenden notwendigen Veränderungen zusammengefasst werden.

4.4.3. Auswertung der Analyse der bestehenden Anwendung

Die vorangegangene Analyse der einzelnen ausgewählten Komponenten hat gezeigt, dass eine Trennung zwischen GUI und Applikationslogik ansatzweise durch eine Trennung in verschiedene Methoden vorliegt. Daher lässt sich die Beschreibung der GUI problemlos aus dem vorhandenen Programmcode entnehmen und durch Textauszeichnung in separaten Dateien ersetzen. Ein Großteil der Applikationslogik kann unverändert beibehalten werden. Die Funktionalität soll nicht verändert werden, lediglich die Herangehensweise. Durch eine Verwendung von MyXaml-Markup wird es voraussichtlich möglich sein, ein gemeinsames Stylesheet für alle Elemente zu definieren. Sicher ist in jedem Fall, dass jedem Element ein individueller Stil zugewiesen werden kann.

In den Entwurf des neuen Systems sollten folgende Punkte miteinbezogen werden:

- Die Klassen `ClientMainForm` und `StartForm` sind sehr ähnlich aufgebaut und unterliegen daher ähnlichen Veränderungen. Es kann jeweils die Methode `InitializeComponent` in ein MyXaml-Dokument ausgelagert werden, da

Alle Rechte vorbehalten

diese fast ausschließlich die Steuerelemente der GUI definiert.

- Bilder liegen in der vorhandenen Version als Ressourcen vor. Bei einer Neugestaltung sollten diese als Dateien in einem Ordner abgelegt sein und möglichst im Stylesheet bzw. in den darin enthaltenen Skins über die Angabe eines Pfades darauf zugegriffen werden.
- Aus dem zuvor genannten Grund sollten die Methoden `getBigIcon` und `getTabIcon` der Schnittstelle `IESGPlugIn` lediglich String-Arrays mit Pfaden zu den benötigten Bildern zurückgeben, statt wie bisher Bitmap-Arrays, welche die vom Plug-In als Ressourcen bereitgestellten Bilder enthalten.
- Im TVZapPlugIn sollten die Angabe von Eigenschaften und Erzeugung der Steuerelemente aus dem Konstruktor bzw. der `Paint`-Methode entfernt und in einem MyXaml-Dokument bzw. dem zugehörigen Stylesheet beschrieben werden. Die `Paint`-Methode ist jedoch weiterhin notwendig zum Zeichnen der Begrenzungslinie zwischen der Liste und dem PInfofenster. Alle Komponenten, die gegenwärtig dynamisch erzeugt werden, könnten auch weiterhin dynamisch erzeugt werden, jedoch in Form von MyXaml-Markup, welches wiederum in das MyXaml-Dokument eingefügt und vom Parser instantiiert wird.
- Das bisherige PInfofenster ist mit identischem Aufbau direkt integriert in alle drei Zapping-Plug-Ins und könnte daher als eine separate Komponente umgesetzt werden, die dann individuell von den einzelnen Plug-Ins aufgerufen wird. Die übergebenen Items werden bereits jeweils auf ihren Typ hin überprüft. Somit wären kaum Änderungen an der Funktionalität notwendig. Es ist jedoch unnötig, dass bei Übergabe eines neuen Items der Fortschrittsbalken und die MessageBox gelöscht und neu erstellt werden. Eine Aktualisierung der angezeigten Werte sollte genügen.
- Für die Instantiierung durch den MyXaml-Parser muss der ImageButton verändert werden. Der leere Konstruktor muss angepasst sowie Getter und Setter für die Fokussierbarkeit und den Fokus-Index geschaffen werden. Die parameterhaltigen Konstruktoren sowie die Klasse `ButtonResources` werden nicht mehr benötigt.

Um das Branding des ESG-Client im Anschluss an die Umsetzung der analysierten notwendigen Änderungen durchführen zu können, soll nachfolgend die spätere Vorgehensweise anhand einiger Anwendungsfälle definiert werden.

Alle Rechte vorbehalten

4.5. Anwendungsfälle

Zur Durchführung des Brandings soll es dem jeweiligen Benutzer möglich sein, Stylesheet-Dokumente und Skins zu erstellen, zu modifizieren oder zu löschen sowie der Anwendung ein vorhandenes Stylesheet und einen darin enthaltenen Skin zuzuweisen. Standardmäßig muss bereits ein Stylesheet mit mindestens einem Skin existieren um die Anwendung benutzen zu können. Benutzer des modifizierten ESG-Client sind, basierend auf der vorangehenden Zielgruppen-Definition, Programmierer, Screendesigner sowie andere Angestellte der TSI bzw. von der TSI beorderte externe Testpersonen mit grundlegenden bis keinen Programmierkenntnissen. Die Akteure in den einzelnen Anwendungsfall-Definitionen sollen daher verallgemeinernd als TSI-Mitarbeiter bezeichnet werden.

Die wichtigsten Schritte zur erstmaligen Durchführung des Brandings des ESG-Clients sind im nachfolgenden Anwendungsfall-Diagramm dargestellt. Das Löschen eines Stylesheets oder Skins wird dabei nicht berücksichtigt.

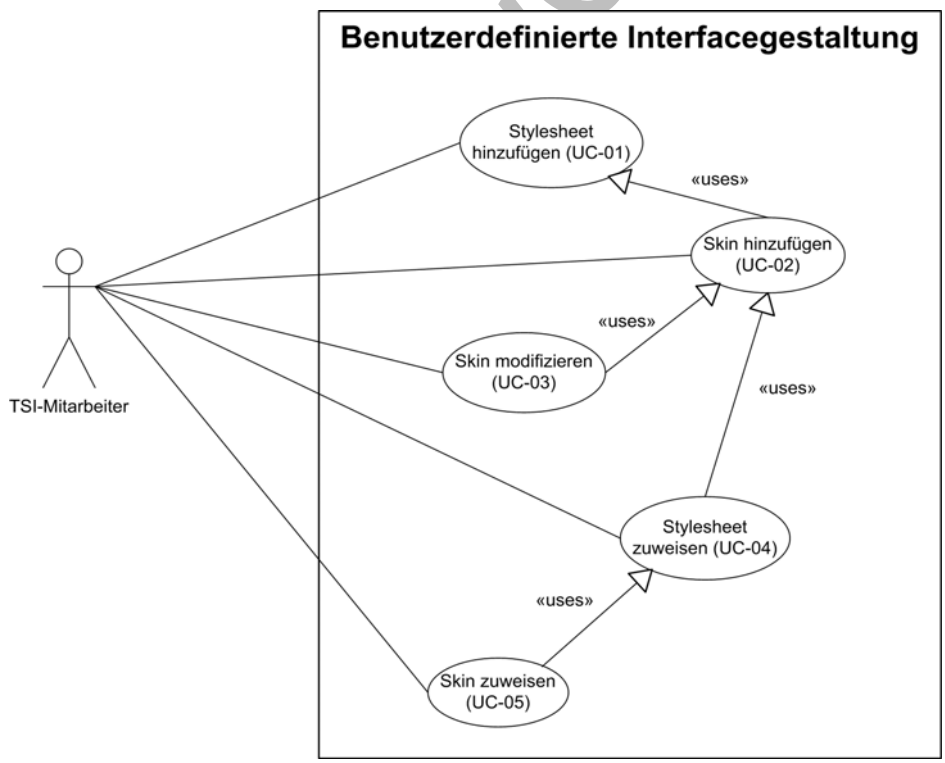

Abbildung 20: Anwendungsfall-Diagramm “Branding“

Jeder Schritt stellt einen Anwendungsfall (engl. use case) dar. Einzelne Schritte können dabei voneinander abhängig sein. Dies wird deutlich gemacht durch eine Use-Beziehung, dargestellt mittels eines Pfeiles mit der Beschriftung <<uses>>. So ist es beispielsweise notwendig, dass zunächst ein Stylesheet-Dokument ausgewählt werden muss, welches mindestens einen Skin

Alle Rechte vorbehalten

enthält, bevor ein Skin zugewiesen werden kann. Der Anwendungsfall „Skin zuweisen“ (UC-05) <<verwendet>> den Anwendungsfall „Stylesheet zuweisen“ (UC-04).

Die in Abbildung 20 dargestellten Anwendungsfälle werden in den nachfolgenden Tabellen detailliert beschrieben:

Tabelle 2: Anwendungsfall 1 – Hinzufügen eines neuen Stylesheet-Dokumentes

UC-01 – Hinzufügen eines neuen Stylesheet-Dokumentes	
Zusammenfassung	Anlegen eines neuen MyXaml-Stylesheet-Dokumentes
Akteur	TSI-Mitarbeiter
Bedingung	-
Ablauf	1. Erstellen eines leeren Dokumentes 2. Einfügen einer XML-Deklaration 3. Einfügen eines MyXaml-Elementes 4. Abspeichern des XML-Dokumentes 5. Fortfahren mit UC-02
Alternative Bedingung	Es ist bereits ein Stylesheet-Dokument vorhanden.
Alternativer Ablauf	1. Erstellen einer Kopie der vorhandenen Datei 2. Umbenennen der Kopie 3. siehe UC-02 oder UC-03 4. siehe UC-05
Notizen und Fragen	• Hinweise zum Erstellen eines neuen Stylesheet-Dokumentes sind im Benutzerhandbuch der modifizierten Anwendung zu finden. • Im alternativen Verlauf sind keinerlei Programmierkenntnisse erforderlich.

Tabelle 3: Anwendungsfall 2 – Hinzufügen eines neuen Skins

UC-02 – Hinzufügen eines neuen Skins	
Zusammenfassung	Beschreibung einer neuen Sammlung von Stilattributen
Akteur	TSI-Mitarbeiter
Bedingung	UC-01
Ablauf	1. Öffnen des Stylesheet-Dokumentes 2. Erzeugen eines MxComponent-Elementes 3. Definieren der Stile 4. Speichern des Dokumentes 5. Schließen des Dokumentes
Alternative Bedingung	Ein Stylesheet-Dokument mit mindestens einem Skin ist bereits vorhanden.

Alle Rechte vorbehalten

UC-02 – Hinzufügen eines neuen Skins	
Alternativer Ablauf	1. Öffnen des Stylesheet-Dokumentes 2. Kopieren eines vorhandenen MxComponent-Elementes inklusive aller darin befindlichen Stildefinitionen 3. Einfügen des MxComponent-Elementes 4. Schrittweise Modifizieren der Stildefinitionen 5. Speichern des Dokumentes 6. Schließen des Dokumentes
Notizen und Fragen	• Zur Definition der Stilattribute steht eine Skinning-Anleitung zur Verfügung, in der alle Steuerelemente, deren Eigenschaften und die zugehörigen möglichen Werte aufgelistet sind. • Im alternativen Verlauf sind keinerlei Programmierkenntnisse erforderlich.

Tabelle 4: Anwendungsfall 3 – Modifizieren eines Skins

UC-03 – Modifizieren eines Skins	
Zusammenfassung	Verändern der Stilattribute eines vorhandenen Skins durch Hinzufügen bzw. Entfernen von Stilattributen oder Ändern von Attributwerten
Akteur	TSI-Mitarbeiter
Bedingung	UC-02
Ablauf	1. Öffnen des Stylesheet-Dokumentes 2. Auswählen des zu modifizierenden Skins 3. Verändern der Stildefinition 4. Speichern des Dokumentes
Alternative Bedingung	-
Alternativer Ablauf	-
Notizen und Fragen	Zur Definition der Stilattribute steht eine Skinning-Anleitung zur Verfügung, in der alle Steuerelemente, deren Eigenschaften und die zugehörigen möglichen Werte aufgelistet sind.

Tabelle 5: Anwendungsfall 4 – Zuweisen eines Stylesheet-Dokumentes

UC-04 – Zuweisen eines Stylesheet-Dokumentes	
Zusammenfassung	Ändern des von der Anwendung verwendeten Stylesheets
Akteur	TSI-Mitarbeiter
Bedingung	UC-01
Ablauf	1. Starten der Anwendung 2. Starten des Skinning-Features 3. Auswählen des Stylesheets aus einer Liste 4. Fortfahren mit UC-05
Alternative Bedingung	-
Alternativer Ablauf	-
Notizen und Fragen	Zur Änderung des verwendeten Stylesheets stehen Hinweise im Benutzerhandbuch der modifizierten Anwendung zur Verfügung.

Alle Rechte vorbehalten

Tabelle 6: Anwendungsfall 5 – Zuweisen eines Skins

UC-05 – Zuweisen eines Skins	
Zusammenfassung	Ändern des von der Anwendung verwendeten Skins
Akteur	TSI-Mitarbeiter
Bedingung	a) UC-02 b) Im Stylesheet-Dokument sind mindestens zwei Skins definiert. c) UC-04
Ablauf	1. Beginn mit UC-04 2. Auswahl eines Skins aus einer Liste 3. Bestätigen der Auswahl durch Drücken des entsprechenden Buttons 4. Schließen der Anwendung 5. Erneutes Starten der Anwendung um die Änderungen sehen zu können
Alternative Bedingung	-
Alternativer Ablauf	-
Notizen und Fragen	Zur Änderung des verwendeten Skins stehen Hinweise im Benutzerhandbuch der modifizierten Anwendung zur Verfügung.

Anhand dieser Definitionen der einzelnen Anwendungsfälle sowie der vorangegangenen Analyse der bestehenden Anwendung lassen sich Anforderungen an das neue System ableiten, auf die im folgenden Kapitel eingegangen werden soll.

4.6. Resultierende Anforderungen an die Anwendung

Die sich aus den zuvor behandelten Abschnitten ergebenden Anforderungen an die Neugestaltung der bestehenden Anwendung sollen nachfolgend zusammengefasst werden.

4.6.1. Allgemeine Anforderungen

Im Rahmen dieser Diplomarbeit sollen die zuvor ausgewählten Komponenten des ESG-Client (siehe 4.4.2.) soweit modifiziert werden, dass eine Anpassung der grafischen Benutzungsschnittstelle der Anwendung unter Verwendung von Stylesheets möglich ist. Das vorliegende Plug-In-Konzept wird beibehalten. Die Implementierung erfolgt mit Hilfe der Entwicklungsumgebung Visual Studio .NET 2003 sowie der Programmiersprache C#. Die resultierende Anwendung soll auf einem Pocket PC 2003 Second Edition lauffähig sein.

Alle Rechte vorbehalten

4.6.2. Anforderungen hinsichtlich der „Skinningfähigkeit“

Dieser Abschnitt umfasst alle Veränderungen, die notwendig sind, um eine benutzerdefinierte Interfacegestaltung des ESG-Client zu ermöglichen. Die einzelnen Punkte sollen nachfolgend jeweils kurz benannt und erläutert werden:

Trennung von GUI und Applikationslogik (essentiell)
Die vorliegende Systemarchitektur wird weitestgehend beibehalten. Es erfolgt jedoch eine Trennung der Definition der GUI von der zugehörigen Applikationslogik innerhalb der einzelnen Klassen der ausgewählten Komponenten.

XML-basierte Beschreibung der GUI-Elemente (essentiell)
Die Beschreibung der Elemente der GUI erfolgt mittels der XML-basierten Markup-„Sprache“ MyXaml. Die Instantiierung und Initialisierung der Steuerelemente sowie die Anbindung der entsprechenden Ereignisse werden vom MyXaml-Parser vorgenommen.

Anpassbarkeit aller GUI-Elemente (zum Teil optional)
Jedem Element der grafischen Benutzungsschnittstelle wird genau ein Stil zugeordnet. Durch Modifikation der darin definierten Stilattribute soll es möglich sein, das Aussehen des betreffenden Elementes hinsichtlich Farben, Texten, Bildern, Schriften und Größen sowie dessen Position auf dem Bildschirm anzupassen. Die TSI stellt in diesem Zusammenhang die Anforderung, dass das Aussehen des ESG-Client schnellstmöglich an verschiedene Anbieter anpassbar sein soll. Als besonders wichtig wird dabei die Austauschbarkeit von Farben und Bildern angesehen. Größen, Positionen und sonstige Eigenschaften der Elemente können zunächst erhalten bleiben. Die Anpassbarkeit von Größe und Position der GUI-Elemente ist daher im Rahmen dieser Arbeit optional.

Entsprechend der Hardware-Spezifikation eines Pocket PC 2003 SE sollten Bilder als Bitmaps mit einer Farbtiefe von maximal 16 Bit vorliegen, da Windows CE zum gegenwärtigen Zeitpunkt lediglich Icons im Bitmap-Format und keine 32 Bit Farbtiefe unterstützt. Die Auflösung der Bilder sollte 72 dpi betragen.

Kompatibilität der Steuerelemente mit MyXaml (essentiell)
Das Interaktionselement ImageButton soll unter Verwendung von Textauszeichnung beschrieben und durch den MyXaml-Parser instantiiert werden können.

Alle Rechte vorbehalten

4.6.3. Anforderungen hinsichtlich der „Skinning-Durchführung“

Das Aussehen der GUI soll so schnell und einfach wie möglich angepasst werden können. Daher wird der bestehende ESG-Client soweit modifiziert, dass die Verwendung von Stylesheets möglich ist. Weiterhin soll ein einfaches Skinning-Feature implementiert werden, mit dessen Hilfe diese Fähigkeit zur Durchführung des Skinnings getestet und demonstriert werden kann.

Das Erstellen der benötigten Stylesheets und zugehörigen Skins wird jedoch vorerst nur für Benutzer, laut Zielgruppen-Definition in diesem Fall TSI-Mitarbeiter, ohne weiteres durchführbar sein, die sich mit der Anwendung bereits beschäftigt haben. Als Hilfestellung soll daher zusätzlich eine Skinning-Anleitung verfasst werden, in der die Vorgehensweise für Benutzer erklärt wird, die unerfahren sind im Umgang mit der Anwendung und/oder keinerlei Programmierkenntnisse besitzen. Diese Kompliziertheit hinsichtlich des Skinnings ergibt sich aus der Tatsache, dass die Stylesheets und Skins ebenfalls in Form von MyXaml-Elementen definiert werden und im Rahmen der vorliegenden Diplomarbeit keine zusätzliche Anwendung entwickelt wird, die eine Erzeugung von Stylesheets über eine GUI ermöglicht. Daher ist es notwendig, dass der Benutzer geringfügige HTML- oder XML-Kenntnisse besitzt, den Aufbau der Anwendung hinsichtlich der Anordnung der Steuerelemente sowie deren Zugehörigkeit zu bestimmten Formularen oder Panels kennt und somit selbstständig Stylesheets und Skins in Form von MyXaml-Markup erzeugen kann.

Basierend auf den ermittelten Anforderungen an den modifizierten ESG-Client, die sich aus einer genauen Analyse der bestehenden Anwendung sowie einer Definition diverser Anwendungsfälle für das zukünftige System ableiten, kann nachfolgend die Aufgabenstellung spezifiziert und ein konkretes Ziel definiert werden, welches im Rahmen der vorliegenden Diplomarbeit erreicht werden soll.

4.7. Zieldefinition

Das Ziel dieser Diplomarbeit besteht darin, die zuvor ausgewählten Komponenten des prototypischen ESG-Client in ihrer Implementierung soweit zu modifizieren, dass eine Trennung von GUI und zugehöriger Applikationslogik vorliegt sowie eine Verwendung von Stylesheets möglich ist. Als Resultat dieser Modifikation sollen die folgenden Komponenten entstehen:

Alle Rechte vorbehalten

- Ein ESG-Client
- Ein Startbildschirm
- Ein Plug-In zur Übersicht aller TV-Services
- Ein separates Panel zur Darstellung der Detailinformationen zu den Services, das ebenso für die Plug-Ins zur Übersicht aller Audio- und Service-Dienste benutzt werden kann
- Ein Stylesheet, das die Stildefinitionen für alle genannten Komponenten beinhaltet
- Ein einfaches Werkzeug zu Test- und Demonstrationszwecken der realisierten Skinningfähigkeit, nachfolgend als *Skinning-Feature* bezeichnet

Die einzelnen Komponenten sollen jeweils aus mindestens zwei Dateien zusammengesetzt sein. Eine Datei beinhaltet die Beschreibung der GUI in Form von MyXaml-Elementen. Eine oder mehrere Dateien enthalten die zugehörigen Klassen und Methoden in Form von C#-Programmcode. Das gemeinsame Stylesheet umfasst, ebenfalls in Form von MyXaml-Markup, die einzelnen Stildefinitionen aller Formulare und deren Steuerelemente. Mit Hilfe einer rudimentären Implementierung eines „Skinning-Features“ soll demonstriert werden, dass die Verwendung unterschiedlicher Skins möglich ist. Dazu sollen verfügbare Stylesheets und darin definierte Skins angezeigt, ausgewählt und die entsprechenden Stile von der Anwendung übernommen werden können.

Basierend auf der in diesem Kapitel durchgeführten Analyse soll im Folgenden ein konkreter Entwurf des neugestalteten ESG-Client angefertigt werden.

Alle Rechte vorbehalten

5. Re-Design

Gegenstand dieses Kapitels ist die Anfertigung eines konkreten Entwurfes zur Modifikation des bestehenden ESG-Clients. Zunächst soll die Systemarchitektur der veränderten Anwendung definiert werden. Im Anschluss daran werden Aussagen zum Aufbau der Stylesheets und Skins getroffen sowie eine Zuordnung individueller Stile zu den verwendeten Steuerelementen vorgenommen.

5.1. Systemarchitektur

Der Entwurf der Systemarchitektur beruht auf den Aussagen, die in der vorhergehenden Analyse des bestehenden ESG-Client getroffen wurden. Die darin angeführten Änderungsvorschläge sollen in den folgenden Abschnitten konkretisiert und für die Realisierung ausgearbeitet werden.

5.1.1. Trennung von GUI und Applikationslogik

Wie bereits in der Zieldefinition festgelegt wurde, soll zur Umsetzung einer Trennung von GUI-Definition und unterliegender Applikationslogik zusätzlich zu den bereits bestehenden Klassen eine Aufteilung der zu modifizierenden Komponenten in verschiedene Dateien erfolgen. Die Beschreibung der GUI-Elemente wird dabei in MyXaml-Dokumente ausgelagert. Die vorhandene Applikationslogik und die an die Interaktionselemente gebundene Eventbehandlung soll weitgehend erhalten bleiben.

Inwiefern eine solche Trennung in den, im Rahmen der Analyse, ausgewählten Komponenten umgesetzt werden soll, zeigt die nachfolgende Abbildung 21:

Alle Rechte vorbehalten

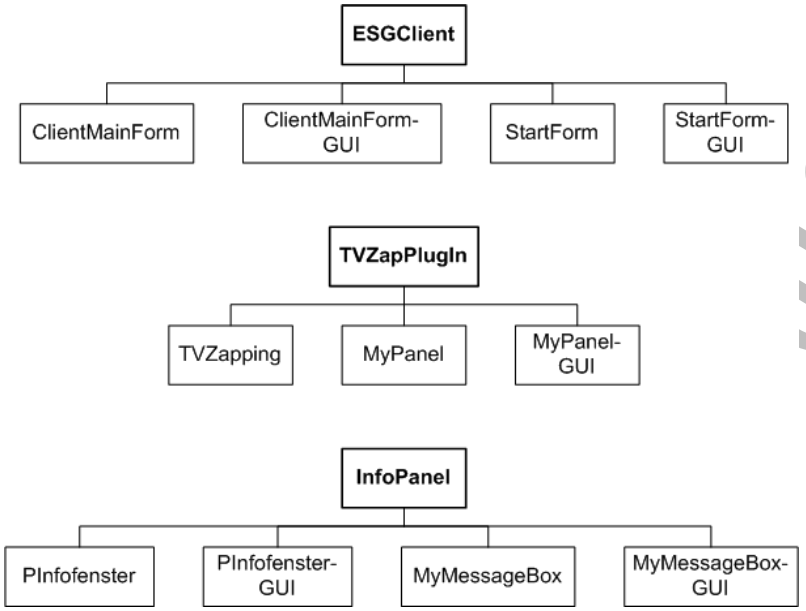

Abbildung 21: Entwurf zur Umsetzung einer GUI-Logik-Trennung

Die im Namensraum `ESGClient` enthaltenen Klassen `ClientMainForm` und `StartForm` werden jeweils in ihre GUI-Beschreibung sowie den zugehörigen C#-Programmcode zerlegt. Es liegen daher weiterhin die zuvor bestehenden zwei Klassen sowie zwei zusätzliche MyXaml-Dokumente vor. Über die Schnittstelle `IESGPlugIn` wird das TVZapPlugIn vom ESG-Client mittels der vorhandenen Funktionen geladen.

Die Klasse `TVZapping`, welche das IESGPlugIn implementiert, bleibt erhalten und wird lediglich in den Methoden an die nachfolgend beschriebenen Veränderungen der Schnittstelle angepasst. Die Klasse `MyPanel`, die das eigentliche Plug-In ausmacht, wird ebenfalls in ihre Darstellungsebene und die unterliegende Logik zerlegt. Die Klassen `MyItemList` und `MyUpdatedItemList` sollen überarbeitet und als Methoden in die Klasse `MyPanel` integriert werden. Die ursprünglichen Klassen entfallen daraufhin.

Die Definition der Klassen `PInfofenster` und `MyMessageBox` erfolgt innerhalb einer separaten Komponente im Namensraum `InfoPanel`. Die GUI-Definitionen beider Klassen werden ebenfalls in MyXaml-Dokumente ausgelagert.

Ein einfaches Beispiel einer GUI-Beschreibung in Form von MyXaml-Elementen wurde bereits in den Grundlagen im Kapitel MyXaml (siehe 3.4.3.) gezeigt. Im Hinblick auf die tatsächliche Umsetzung der GUI-Logik-Trennung unter Verwendung von Stylesheets ist in Abbildung 22 ein konkretes Beispiel für die Umsetzung des StartForm zu sehen. Dargestellt sind die Beschreibungen des Formulares selbst sowie der Elemente zur Repräsentation der Titelzeile und des Druckschalters zum Beenden der Anwendung.

Alle Rechte vorbehalten

```
 1| <?xml version="1.0" encoding="utf-8"?>
 2| <MyXaml xml:lang="en-US"
 3|         xmlns:def="Definition"
 4|         xmlns:ref="Reference"
 5|         xmlns="System.Windows.Forms"
 6|         xmlns:opwf="OpenNETCF.Windows.Forms">
 7|
 8|  <Form ref:Name="StartForm" MxStyle="{StartFormStyle}">
 9|
10|   <Include Src="Stylesheet" ComponentName="Skin" Postfix="1"/>
11|
12|   <Controls>
13|    <PictureBox ref:Name="TitleBar" MxStyle="{TitleBarStyle}">
14|     <Image>
15|      <Bitmap File="titlebar.bmp"/>
16|     </Image>
17|    </PictureBox>
18|    <opwf:ButtonEx ref:Name="Exit" MxStyle="{ExitStyle}" Click="Exit_Click">
19|     <opwf:BackgroundImage>
20|      <Bitmap File="exit.bmp"/>
21|     </opwf:BackgroundImage>
22|    </opwf:ButtonEx>
23|   </Controls>
24|
25|  </Form>
26| </MyXaml>
```

Abbildung 22: Beispiel für ein MyXaml-Dokument zur Beschreibung der StartForm-GUI

Wie im obigen MyXaml-Dokument zu erkennen ist, können Komponenten von Drittanbietern problemlos verwendet werden. Dabei handelt es sich in Abbildung 22 um einen ButtonEx (Zeile 18), der im OpenNETCF SmartDeviceFramework[22] im Namensraum `OpenNETCF.Windows.Forms` definiert ist. Durch Angabe des entsprechenden Namensraumes sowie der Beschreibung des Buttons mittels der Namen der zugehörigen Klassen, Eigenschaften und Eventhandler in Form von Elementen und Attributen, ist eine Instantiierung durch den MyXaml-Parser ohne weiteres möglich. Ebenso wie der im Beispiel verwendete ButtonEx soll auch der für den ESG-Client entwickelte ImageButton beschrieben werden können. Die dazu notwendigen Modifikationen an der entsprechenden Klasse werden im nächsten Abschnitt behandelt.

5.1.2. Anpassung des ImageButtons

Um den ImageButton auf ähnliche Art und Weise mit MyXaml-Markup beschreiben und durch den Parser instantiieren lassen zu können, wie den in Abbildung 22 dargestellten ButtonEx, muss zunächst ein Namensraum definiert werden. Als Namensraum wurde der

22 Sammlung von OpenSource-Steuerelementen, die aufgrund der begrenzten Anzahl vorhandener Steuerelemente speziell zur Verwendung auf dem .NET Compact Framework entwickelt wurden. Siehe http://www.opennetcf.org/CategoryView.aspx?category=Home

Alle Rechte

gegenwärtige Assemblyname `ImageButton` festgelegt. Die Abbildung des Namensraumes nach MyXaml sowie die Beschreibung eines ImageButtons sind in Abbildung 23 zu sehen:

```
 1| <!-- Namespace-Mapping -->
 2| xmlns:imgb="ImageButton"
 3|
 4| <!-- Button-Beschreibung -->
 5| <imgb:ImageButton def:Name="NachOben"
 6|                   Size="50, 20"
 7|                   Location="10, 130"
 8|                   Focusable="false"
 9|                   ButtonClicked="imgb_ButtonClicked">
10|  <imgb:NormalImage>
11|   <Bitmap File="btnUp.bmp"/>
12|  </imgb:NormalImage>
13|  <imgb:PressedImage>
14|   <Bitmap File="btnUp_pressed.bmp"/>
15|  </imgb:PressedImage>
16| </imgb:ImageButton>
```

Abbildung 23: Beispiel für die MyXaml-Beschreibung eines ImageButtons

Generell ist, abgesehen von der Eigenschaft `Focusable` (Zeile 8), eine solche Beschreibung bereits ohne eine Modifikation der Klasse `ImageButton` möglich. Der ImageButton kann jedoch, wie die Analyse der Komponente ergeben hat, nicht auf dieselbe Art und Weise wie im bestehenden ESG-Client verwendet werden. Eine Anpassung des parameterlosen Konstruktors, die Ergänzung der Eigenschaften `Focusable` und `FocusIndex` sowie eine Überprüfung der Zustandsbilder in der `OnPaint`-Methode sind notwendig, damit die bisherige Funktionsweise ohne Einbußen beibehalten werden kann.

Im leeren Konstruktor muss bei der Instantiierung des Buttons standardmäßig der Normalzustand angenommen und somit die Eigenschaften `isPushed` und `isPressed` auf den Wert `false` gesetzt werden, wie es in den bisher verwendeten Konstruktoren mit Parameterliste der Fall war. Da die Fokussierbarkeit eines ImageButtons ab sofort nicht mehr automatisch, abhängig vom verwendeten Konstruktor auf `true` oder `false` festgesetzt wird, sondern manuell festgelegt werden soll, muss zusätzlich im leeren Konstruktor der Wert für die Fokussierbarkeit standardmäßig auf den Wert `false` gesetzt werden.

Zum manuellen Setzen der Fokussierbarkeit müssen eine Set- und eine Get-Methode definiert werden, über die die Eigenschaft `Focusable` auf `true` oder `false` gesetzt bzw. deren aktueller Wert abgerufen werden kann. Diese Veränderung ist notwendig, da das TVZapPlugIn nicht fokussierbare ImageButtons, das InfoPanel hingegen fokussierbare ImageButtons benötigt. Der Wert der Fokussierbarkeit legt dabei fest, ob nach dem Druck auf

Alle Rechte vo

einen ImageButton jener fokussiert bleibt oder der Fokus an ein anderes Steuerelement weitergegeben wird.

Das manuelle Setzen des Fokus-Index ist notwendig, damit wie bisher im InfoPanel problemlos zwischen den ImageButtons `Buy`, `Info` und `Start` navigiert werden kann. Diese sind gegenwärtig in einer ArrayList enthalten und mit einem horizontalen Fokus-Index versehen, der es ermöglicht, durch Angabe des jeweiligen Integer-Wertes den Fokus an den Button rechts oder links vom aktuell fokussierten Button zu übergeben. Da der Fokus-Index ebenso wie die Fokussierbarkeit ausschließlich unter Verwendung des entsprechenden Konstruktors über Parameter gesetzt werden konnte, muss diese Eigenschaft zukünftig separat definiert werden können.

Zuletzt muss eine Modifikation der `OnPaint`-Methode durchgeführt werden, da diese lediglich die Bilder für die einzelnen Button-Zustände zuweist, jedoch nicht überprüft, ob ein Bild gesetzt wurde oder nicht. Diese Prüfung wurde in der aktuellen Implementierung des ImageButtons ebenfalls in einem parameterhaltigen Konstruktor vorgenommen. Daher muss in der Anpassung der Klasse `ImageButton` in der `OnPaint`-Methode zunächst geprüft werden, ob ein Bild für einen bestimmten Zustand vorhanden ist oder nicht. Ist kein Bild vorhanden, so muss das Bild des Normalzustandes für den jeweiligen Zustand angezeigt werden. Ist dieses nicht definiert, soll eine Fehlermeldung ausgegeben werden.

Mit den genannten Änderungen sollte es möglich sein, den ImageButton so zu beschreiben, wie es in Abbildung 23 gezeigt wurde. Die Klasse `ButtonResources` wird zukünftig nicht mehr benötigt.

5.1.3. Anpassung der Plug-In-Schnittstelle

Damit die Durchführung eines Brandings möglich ist, müssen, wie während der Analyse bereits festgestellt, alle verwendeten Bilder direkt als Dateien in einem Ordner auf dem Endgerät bzw. dem Pocket PC-Emulator vorliegen, anstatt wie in der aktuellen Version des ESG-Client als Ressourcen aus den Klassenbibliotheken der Anwendung geladen zu werden. Dies bedeutet, dass alle von den Plug-Ins benötigten Bilder nicht mehr von diesen selbst sondern vom aufrufenden Client bereitgestellt werden müssen. Daher ist eine Modifikation in der Programmierung der Schnittstelle `IESGPlugIn` erforderlich.

In der gegenwärtig vorliegenden Version werden die zur Repräsentation eines Plug-Ins im ESG-Client benötigten Bilder aus den Ressourcen des Plug-Ins ausgelesen und über die

Alle Rechte vorbehalten

aufgerufenen Methoden `getBigIcon` und `getTabIcon` in Form von Arrays an den ESG-Client übergeben. Da die geforderten Bilddateien in der modifizierten Version bereits in der Anwendung selbst vorliegen, ist es für den ESG-Client lediglich notwendig, die Pfade zu den zu verwendenden Bildern zu kennen. Daher sollen die zuvor genannten Methoden anstelle von Bitmap-Arrays zukünftig String-Arrays zurückgeben, welche die benötigten Pfadangaben zu den entsprechenden Bildern enthalten, die aus den betreffenden Plug-Ins ausgelesen wurden. Somit muss beispielsweise die Methode `getBigIcon` statt wie bisher `Bitmap[] getBigIcon()`folgendermaßen definiert sein: `string[] getBigIcon()`. Die Methode `getSmallIcon` ist zwar in der ursprünglichen Version der Schnittstelle implementiert, wurde jedoch nicht vom ESG-Client verwendet und kann demzufolge in der neuen Version aus der Schnittstelle entfernt werden.

5.1.4. Package-Diagramm der modifizierten Anwendung

Anhand der in den vorhergehenden Abschnitten ausgearbeiteten Modifikationen lässt sich die Systemarchitektur der neu gestalteten Anwendung in Form des folgenden Package-Diagramms veranschaulichen:

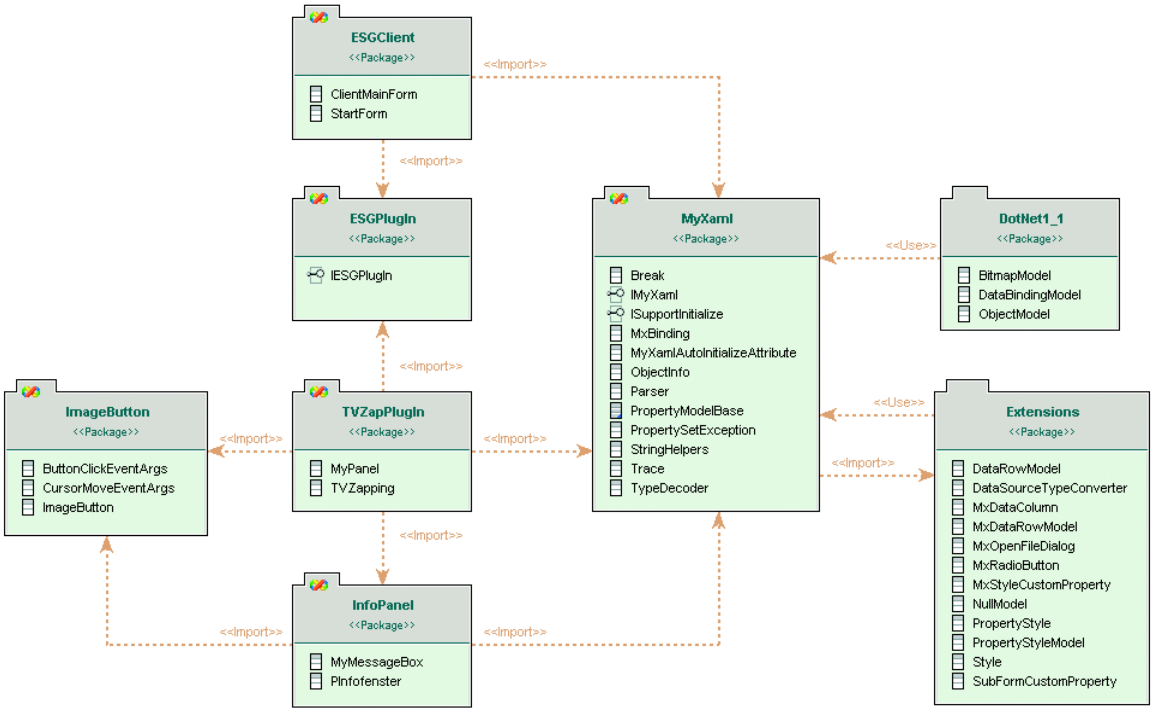

Abbildung 24: Package-Diagramm des modifizierten ESG-Client (Entwurf)

Der ESG-Client ruft über die Schnittstelle `IESGPlugIn` das TVZapPlugIn auf. Dieses bindet in Zukunft das PInfofenster in Form der Komponente InfoPanel ein. Beide verwenden

vorbehalten

den modifizierten ImageButton. Neu hinzu gekommen sind das MyXaml-Package und die zugehörigen Packages DotNet1_1 und Extensions. Das Zusammenspiel dieser drei Komponenten ermöglicht die Instantiierung der MyXaml-Beschreibungen der GUI-Elemente sowie die Verwendung von Stilattributen.

Um die Skinningfähigkeit der Anwendung testen zu können, welche mit den zuvor genannten Modifikationen erzielt werden soll, ist es notwendig, zusätzlich ein simples Skinning-Feature zu entwickeln, mit dessen Hilfe die Anpassung der grafischen Benutzungsschnittstelle des ESG-Client schnell und einfach durchgeführt werden kann. Der Entwurf dieser Funktionalität ist Gegenstand des nächsten Abschnittes.

5.1.5. Entwurf des Skinning-Features

Mit Hilfe des Skinning-Features soll nach der Modifikation des ESG-Client getestet und demonstriert werden, inwieweit die Umsetzung der angestrebten Skinningfähigkeit realisiert werden konnte. Zu diesem Zweck ist es notwendig, mit Hilfe dieses Werkzeuges ein Stylesheet auswählen zu können sowie zu ermitteln, welche Skins darin definiert sind und einen dieser Skins der Anwendung zuzuweisen. Dazu genügt im Rahmen dieser Arbeit ein einfaches Formular, auf dem die zur Verfügung stehenden Stylesheets und zugehörigen Skins aufgelistet sind. Für den professionellen Einsatz zur Durchführung des Brandings für eine Vielzahl von Firmen kann auf Grundlage dieses einfachen Werkzeuges eine separate Anwendung mit einem wesentlich größeren Funktionsumfang entwickelt werden.

Abbildung 25 zeigt den Entwurf für das in dieser Arbeit zu erstellende Skinning-Feature:

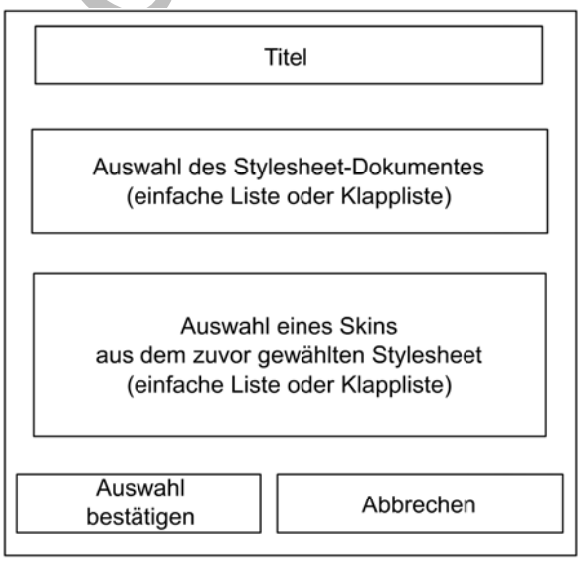

Abbildung 25: Entwurf eines einfachen Skinning-Features

Alle Rechte vorbehalten

Da das Formular lediglich zu Test- bzw. Demonstrationszwecken dienen soll, wird es als verstecktes Feature in den modifizierten ESG-Client integriert. Geöffnet werden kann dieses über einen unsichtbaren Button. Der Einfachheit halber sollen das gewählte Stylesheet und der darin enthaltene Skin vermerkt und erst beim nächsten Start der Anwendung benutzt werden. Wie der Aufbau eines Stylesheets genau aussieht, wird im nachfolgenden Abschnitt festgelegt.

5.2. Stylesheets und Skins

Die folgende Abbildung zeigt den Entwurf eines Stylesheets, in welchem der in Abbildung 22 (siehe 5.1.1.) verwendete Skin definiert wird:

```
 1| <?xml version="1.0" encoding="utf-8"?>
 2| <MyXaml xml:lang="en-US"
 3|         xmlns:def="Definition"
 4|         xmlns="System.Windows.Forms">
 5|
 6|  <MxComponent Name="Skin">
 7|   <Style def:Name="StartFormStyle">
 8|    <StyleProperties>
 9|     <PropertyStyle ClientSize="320, 240" WindowState="Maximized"/>
10|    </StyleProperties>
11|   </Style>
12|   <Style def:Name="TitleBarStyle">
13|    <StyleProperties>
14|     <PropertyStyle Location="0, 0" Size="280, 20"/>
15|     </StyleProperties>
16|   </Style>
17|   <Style def:Name="ExitStyle">
18|    <StyleProperties>
19|     <PropertyStyle Location="280, 0" Size="40, 30"/>
20|    </StyleProperties>
21|   </Style>
22|  </MxComponent>
23|
24| </MyXaml>
```

Abbildung 26: Entwurf eines MyXaml-Stylesheet-Dokumentes mit einem Skin

Wie zu erkennen, ist ein Stylesheet auf ähnliche Art und Weise wie ein in den Grundlagen beschriebenes typisches MyXaml-Dokument aufgebaut (siehe 3.4.3.). In Abbildung 22 wurde per `Include` das im obigen Beispiel definierte `MxComponent`-Element `Skin` eingebunden (Zeile 10). Ein MxComponent kann, wie zu erkennen, abgesehen von Steuerelementen ebenso gut ausschließlich Stildefinitionen enthalten. Die Verwendung von MxComponent-Elementen ermöglicht aus diesem Grund die Auslagerung einzelner Stile und somit die Erstellung eines MyXaml-Stylesheet-Dokumentes. Ein MxComponent-Element kann beliebig viele Stildefinitionen enthalten und deshalb eine gemeinsame Stilsammlung – einen Skin – für den

gesamten ESG-Client darstellen. Demzufolge kann jedes einzelne Steuerelement der Anwendung mit einem individuellen Stil versehen werden. Ein MyXaml-Dokument wiederum kann beliebig viele MxComponent-Elemente enthalten. Daher können in einem Stylesheet-Dokument mehrere Skins beschrieben werden.

Dem StartForm wurde in Abbildung 22 die Stildefinition `StartFormStyle` zugewiesen (Zeile 8). Wie in Abbildung 26 zu erkennen, legt dieser Stil beispielsweise den Clientbereich sowie die Größe des Formular-Fensters fest (Zeile 9). Für die Titelzeile und den Button zum Beenden der Anwendung werden jeweils die Größe und Position auf dem Bildschirm angegeben. In jeder Stildefinition können ausschließlich die spezifischen Eigenschaften des zugehörigen Steuerelementes in Form von Attributen beschrieben werden.

Jedem Skin, wie auch jedem einzelnen Stil, wird ein individueller Name zugewiesen, anhand dessen der Parser feststellen kann, welcher Stil von welchem Element verwendet wird. Im Hinblick auf eine vereinfachte Implementierung sollen an dieser Stelle den einzelnen GUI-Elementen bereits Stilnamen zugewiesen werden. Diese Zuweisung wird mit Hilfe von Screenshots der einzelnen Ansichten der Anwendung vorgenommen, um alle verwendeten Steuerelemente identifizieren zu können. Die entsprechend bearbeiteten Screenshots sind im Anhang A zu finden.

Auf Basis der entworfenen Klassenstruktur soll nachfolgend die zu realisierende Funktionalität der zukünftigen Anwendung erläutert werden.

5.3. Funktionalität

Die bereits bestehende Funktionsweise des ESG-Client soll hinsichtlich der Bedienung der Anwendung unverändert erhalten bleiben und daher an dieser Stelle kurz beschrieben werden:

Beim Start der Anwendung wird der Startbildschirm gezeigt, der die verfügbaren Plug-Ins in Form von Buttons repräsentiert. Beim Klick auf einen Button wird das Plug-In in das im Hintergrund bereits geöffnete ClientMainForm geladen und dieses angezeigt. Der Startbildschirm wird dabei vorübergehend ausgeblendet. Der Benutzer kann in Interaktion mit der Anwendung die vom Plug-In bereitgestellten Funktionen benutzen und somit alle verfügbaren TV-Services übersichtlich in einer Liste angezeigt bekommen, die Detail-informationen zu einem ausgewählten Service einsehen sowie den ausgewählten Service kaufen oder in einer externen Anwendung starten. Beim Schließen des ClientMainForms wird der Startbildschirm erneut eingeblendet, um ein versehentliches Verlassen der Anwendung zu

Alle Rechte vorbehalten

verhindern. Beim Druck des Schließen-Buttons auf dem Startbildschirm wird der ESG-Client endgültig beendet.

Um diese Funktionsweise zu erhalten, soll soweit wie möglich ein Großteil der im ESG-Client verwendeten Methoden übernommen werden. So steht beispielsweise das Laden der Plug-Ins nicht direkt im Zusammenhang mit den Veränderungen der GUI-Erstellung und kann daher in der modifizierten Version vollständig übernommen werden. Es wird lediglich eine minimale Veränderung beim Aktivieren bzw. Deaktivieren der Tabs notwendig sein, um die zukünftig in Form von Pfadangaben übergebenen Bilder zu laden.

Aufwändigere Änderungen in der neuen Version des ESG-Client sind notwendig beim Starten der Anwendung und beim Erzeugen der Plug-In-Panels. So soll zunächst im Konstruktoraufruf des ClientMainForms ein `Parser`-Objekt erzeugt werden. Die Plug-Ins werden mittels der gegenwärtig verwendeten Methode geladen. Die zugehörigen Buttons werden zukünftig in Form von MyXaml-Markup dynamisch in das Dokument eingefügt, welches die Beschreibung des Formulares enthält. Dieses wiederum wird daraufhin vom Parser gelesen, die benötigten Klassen instantiiert und die Anwendung gestartet. Gleiches gilt für die Erzeugung des Startbildschirms. Diesem wird weiterhin das ClientMainForm als Parameter übergeben, um auf die Methoden zum Auslesen der verfügbaren Plug-Ins zugreifen und das ClientMainForm aufrufen zu können. Das Formular selbst wird jedoch auch von einem `Parser`-Objekt aus einer GUI-Beschreibung in Form von MyXaml-Markup erzeugt. Die im MyXaml-Dokument in Form von Attributen übergebenen Methoden zur Eventbehandlung sollen direkt im Programmcode der Anwendung vorliegen, damit nicht zusätzlich zu den Plug-Ins noch weitere Klassenbibliotheken eingebunden werden müssen.

Zum Einbinden eines Plug-Ins nach einem Klick auf einen Button des Startbildschirms oder der Tableiste wird auch zukünftig die Methode `displayPlugIn` aufgerufen, jedoch wird das geforderte Panel im Plug-In auf eine andere Art und Weise erzeugt. Es soll zukünftig ein weiteres `Parser`-Objekt erzeugt werden, dass die GUI-Beschreibung des jeweiligen Panels liest und die Klassen instantiiert. Das resultierende `Panel`-Objekt wiederum kann, wie es in der ursprünglichen Version des ESG-Client der Fall ist, vom ClientMainForm gegen das vorhandene Platzhalter-Panel ausgetauscht werden. Beim Klick auf einen Button des Panels wird das entsprechende Ereignis aus der Plug-In-Klassenbibliothek aufgerufen. Das vom Plug-In verwendete InfoPanel und die zugehörige MessageBox werden auf dieselbe Art und Weise erzeugt. Beim Klick auf einen Button des PInfofensters bzw. der MessageBox werden die zugehörigen Events aus der Assembly InfoPanel.dll aufgerufen.

Alle Rechte vorbehalten

Die beschriebene Funktionsweise soll mittels der folgenden Abbildung veranschaulicht werden:

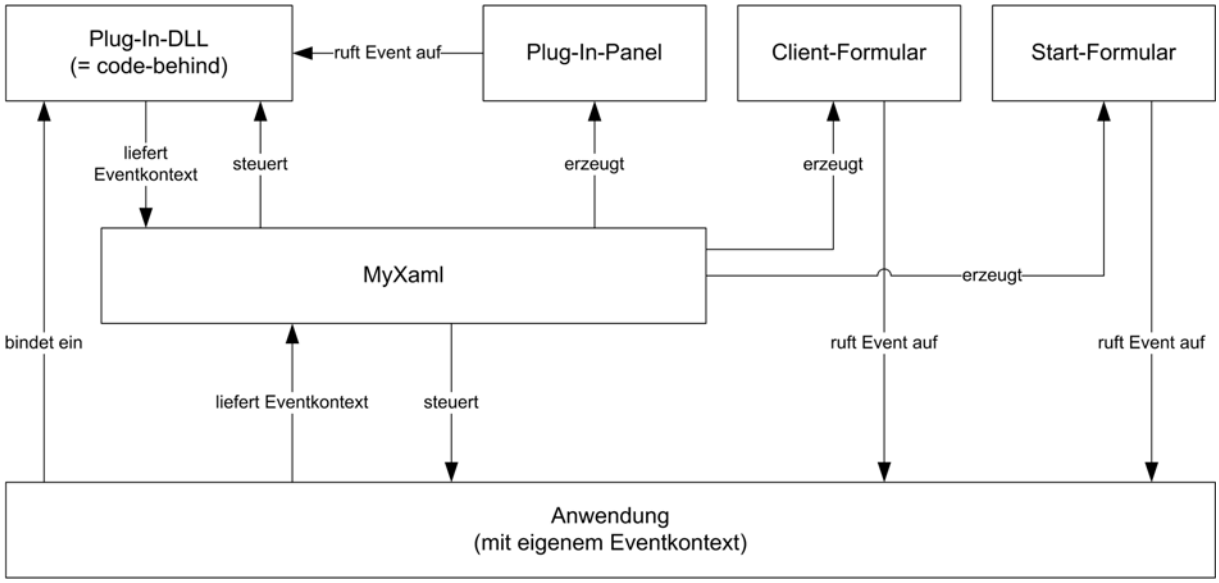

Abbildung 27: Geplante Funktionsweise des modifizierten ESG-Client (vgl. [myxaml7])

Um die beim Erzeugen der Steuerelemente übergebenen Stile verändern und somit das geforderte Branding testweise durchführen zu können, wird nach der Erstellung verschiedener Skins deren Verwendbarkeit mit Hilfe des Skinning-Features geprüft. Dieses soll zunächst hinter dem FHTW-Logo auf dem ClientMainForm versteckt werden. Beim Start dieses Werkzeuges werden alle verfügbaren Stylesheets ermittelt. Nach der Auswahl eines Stylesheets wird dieses nach Skins, definiert in Form von `MxComponent`-Elementen, durchsucht und die gefundenen Skins anhand ihrer Namen aufgelistet. Nach der Auswahl eines Skins werden der alte und neue Stylesheet- und Skin-Name vermerkt und nach einem Neustart der neue Skin verwendet. Dazu werden in allen MyXaml-Dokumenten die Namen des zuvor verwendeten Stylesheets bzw. Skins einfach gegen die neuen Namen ausgetauscht. Die entsprechenden Namen müssen daher jeweils als zusätzliche Parameter beim Aufruf des Startscreens, Plug-Ins oder InfoPanels übergeben werden.

Der in diesem Kapitel angefertigte Entwurf dient als Grundlage für die Realisierung der geforderten Skinningfähigkeit und Durchführung des Skinnings unter Verwendung des entworfenen Werkzeuges und soll im Folgenden umgesetzt werden. Die Vorgehensweise sowie Probleme bei der Implementierung werden im anschließenden Kapitel beschrieben.

Alle Rechte vo

6. Implementierung

In diesem Kapitel soll die Umsetzung des im vorigen Kapitel, auf Basis der ermittelten Anforderungen, angefertigten Entwurfes beschrieben werden. Dabei wird zunächst die Vorgehensweise bei der Implementierung erläutert. Im Anschluss daran werden wesentliche aufgetretene Probleme geschildert und die resultierende Anwendung getestet.

6.1. Vorgehensweise

Die Umsetzung des ESG-Client erfolgte Komponente für Komponente. Dabei wurden alle vorhandenen Einzelprojekte zu einer gemeinsamen Projektmappe `ESGClient` in Visual Studio .NET 2003 zusammengefügt. Die Implementierung lässt sich dabei in die im Folgenden erläuterten zwei Phasen, Umsetzung der Skinninfähigkeit und Durchführbarkeit des Skinnings, einteilen.

6.1.1. Phase 1 – Umsetzung der Skinningfähigkeit

Das Ziel der ersten Phase der Implementierung war es, zunächst die bestehende Anwendung so zu modifizieren, dass diese prinzipiell skinningfähig ist, jedoch bei der Benutzung kein Unterschied in Darstellung und Funktionsweise erkennbar sein soll. Begonnen wurde die Realisierung der Skinningfähigkeit mit der Anpassung des ClientMainForm.

Modifikation des ClientMainForms

Die Klasse wurde in der veränderten Version mit dem Namen `LoadESGClient` versehen um die Funktionsweise als Einstiegspunkt der Anwendung kenntlich zu machen. Bei der Umsetzung sollte im ersten Schritt ausschließlich die GUI-Beschreibung des Formulares ausgelagert und die bisherige Funktionalität durch Übernahme des bestehenden Programmcodes beibehalten werden. Demzufolge wurde zunächst das zugehörige MyXaml-Dokument mit allen benötigten statischen Steuerelementen und den im Entwurf zugeordneten Stilen angelegt. Die Eigenschaften wurden in ein Stylesheet-Dokument ausgelagert und der auf diese Weise erzeugte Skin über den Include-Tag in das zuvor angelegte MyXaml-Dokument eingebunden. Das MyXaml-Dokument muss als Inhalt in das Visual Studio-Projekt integriert werden, damit dieses bei der Weitergabe an den Emulator auf dem Gerät als Datei vorliegt und bei laufender Anwendung dynamisch Elemente darin eingefügt werden können, bevor das Formular und die Steuerelemente erzeugt und dem Benutzer präsentiert werden.

Alle Rechte vorbehalten

Das Stylesheet-Dokument soll ebenfalls auf dem Gerät vorliegen, damit dieses später auch von anderen Komponenten verwendet werden kann. Der Startbildschirm wurde zur Modifikation des ClientMainForms zwecks Zeitersparnis und Fehlervermeidung auskommentiert. Die originalen Plug-Ins wurden zur Kontrolle der automatischen Generierung der Tableiste weiterhin verwendet.

Im folgenden Schritt wurden die bisher dynamisch erzeugten Buttons der Tableiste zunächst in Form von MyXaml-Markup in das Dokument geschrieben, mit den zugehörigen Stildefinitionen versehen und somit die Buttons vom Parser erzeugt. Im Gegensatz zu den dynamisch generierten Steuerelementen werden alle statischen Steuerelemente direkt im Programmcode erzeugt, zur Definitionsliste des Parsers hinzugefügt und von diesem mit den im MyXaml-Dokument beschriebenen Eigenschaften initialisiert. Um die dynamisch erzeugten Button-Elemente in das MyXaml-Dokument einfügen zu können, ist im Dokument ein Platzhalter-Text enthalten. Das Dokument wird vollständig als eine Zeichenkette eingelesen, der entsprechende Platzhalter-Text gegen den String von dynamisch erzeugten Button-Elementen ersetzt und der Inhalt des Dokumentes vollständig gegen die veränderte Zeichenkette ausgetauscht.

Das modifizierte Dokument wird beim Start der Anwendung unter Verwendung der Methode `LoadForm` des Parsers eingelesen, die zu den Tags gehörenden Klassen instantiiert und das Formular schließlich angezeigt. Die Methode `LoadForm` wird an dieser Stelle folgendermaßen verwendet:

```
// LoadForm("Pfad zum MyXaml-Dokument", "NameDesFormulares", Ziel, Parent)
(Form)this.parser.LoadForm(myxamlfile, "clientMainForm", this, null);
```

Der `LoadForm`-Methode wird als Parameter neben dem Pfad zum MyXaml-Dokument, gespeichert in der Variable `myxamlfile`, ebenso als Name des Formulares die Zeichenkette `clientMainForm` übergeben. Als Ziel wird die Klasse `LoadESGClient` selbst, daher an dieser Stelle `this`, angegeben und somit eine Instanz der Klasse erzeugt. Das resultierende Formular ist keinem `Parent`-Objekt zugeordnet. Als Ergebnis liefert der Parser ein Objekt vom Typ `object`, da die Klasse `LoadESGClient` jedoch direkt von der Klasse `System.Windows.Forms.Form` erbt, kann das erzeugte Objekt ohne weitere Prüfung zum Typ Formular gecastet werden und beim Starten der Anwendung angezeigt werden.

Modifikation des StartForms

Im Anschluss an die Modifikation des ClientMainForms erfolgte die Anpassung des StartForms, dessen Instantiierung zu diesem Zweck wieder in das ClientMainForm integriert

Alle Rechte vorbehalten

wurde. Die zunächst begonnene Überlegung, die GUI-Beschreibung des StartForms ebenfalls vom Parser des ClientMainForms einlesen zu lassen wurde wieder verworfen, da zum einen der Startbildschirm, ebenso wie die Plug-Ins, auch relativ leicht austauschbar sein sollte und dies zum anderen weitere Änderungen mit sich gebracht hätte, da keine Möglichkeit bestand dem StartForm bei dessen Instantiierung das `LoadESGClient`-Objekt als Parameter zu übergeben. Ein Versuch der Übergabe mittels einer Set-Methode in der `StartForm`-Klasse, um diese eventuell als code-behind-Instanz verwenden zu können, beendete die Anwendung noch während des Startens auf dem Emulator aufgrund von vermeintlich zu geringem Arbeitsspeicher. Daher sind Aufbau, Aufruf und Funktionsweise unverändert geblieben. Die Skinningfähigkeit des StartForms wurde auf die gleiche Weise realisiert wie zuvor für das ClientMainForm beschrieben. Es musste ebenso ein MyXaml-Dokument angelegt, Platzhalter-Text gegen dynamisch erzeugte Button-Elemente ersetzt und das Dokument von einem `Parser`-Objekt mittels der Methode `LoadForm` verarbeitet werden. Wird ein Button gedrückt, so wird das ClientMainForm angezeigt. Zur Aktivierung bzw. Deaktivierung der Tabs sowie zur Anzeige der Plug-Ins werden die ursprünglichen Methoden des übergebenen `LoadESGClient`-Objektes aufgerufen.

Modifikation des IESGPlugIn

Die einzig notwendige Veränderung der Schnittstelle `IESGPlugIn` betraf, wie bereits im Entwurf (siehe 5.1.3.) erwähnt, die Methoden `getTabIcon` und `getBigIcon`. Es wurde jeweils der Rückgabetyp von einem Bitmap-Array zu einem Array von Strings geändert, so dass die zur Repräsentation der Plug-Ins benötigten Bilder in Form von Pfadangaben an den Client übergeben und von diesem am entsprechenden Ort abgerufen werden können. Die Klassenbibliothek ESGPlugIn.dll wurde nach abgeschlossener Modifikation neu erstellt. Dies zog eine Reihe von Fehlermeldungen im ESGClient-Projekt nach sich, da alle eingebundenen Plug-Ins die Methoden mit dem falschen Rückgabetyp implementierten. Es folgte daher die Anpassung des TVZapPlugIns.

Modifikation des TVZapPlugIns

Zunächst wurden alle weiteren Plug-Ins aus der Projektmappe entfernt, damit das in den Anforderungen geforderte TVZapPlugIn umgesetzt werden konnte. Um Fehlermeldungen beim Laden der Plug-Ins zu verhindern, wurde in der Klasse `LoadESGClient` die Zeichenkette mit den einzubindenden Plug-Ins entsprechend gekürzt.

Die Anpassung des Plug-Ins an die veränderte Schnittstelle wurde zuerst vorgenommen. Dazu wurde die nicht mehr benötigte Methode `getSmallIcon` aus der

Alle Rechte vorbehalten

Klasse `TVZapping` entfernt und die Rückgabetypen der beiden anderen Methoden zu String-Arrays geändert. Die in diesen Methoden verwendete Ladefunktion wurde gegen die Angabe der zu den Bildern gehörenden Pfadangaben ausgetauscht. Alle Bilder und sonstige Dokumente und Dateien werden auf dem Emulator automatisch im Ordner „Program Files/ESGClient“ abgelegt. Damit es möglich ist, die Bilder anhand der entsprechenden Pfadangaben vom Client laden zu lassen, müssen diese in der Projektmappe ESGClient als Inhalt eingebunden werden. Somit werden diese bei der Weitergabe der Anwendung an den Emulator auf das Gerät kopiert und der aufrufende Client kann die Bilder anhand der erhaltenen Pfade im zuvor genannten Ordner vorfinden und auf diese zugreifen. Es ist nicht mehr nötig, dass das Plug-In die Bilder selbst in Form von eingebetteten Ressourcen bereitstellt. Die im TVZapPlugIn-Projekt enthaltenen Bildressourcen können daher entfernt werden.

Im nächsten Schritt wurde die GUI-Beschreibung für das Panel erzeugt. Diese beschränkte sich auf das MyXaml-Element, die Beschreibung und Stilzuweisung des Panels sowie das Einbinden des Stylesheet-Dokumentes. Anstelle der Interaktionselemente wurde in das MyXaml-Dokument jedoch lediglich ein Platzhalter-Text eingefügt, da diese dynamisch, in Abhängigkeit von den verfügbaren VideoItems, zur Laufzeit des Programms hinzugefügt werden. Die Stile mit den Eigenschaften für alle Steuerelemente wurden im Stylesheet ergänzt. Der Konstruktor der Klasse MyPanel wurde dahingehend angepasst, dass zunächst ermittelt wird, ob VideoItems zur Verfügung stehen oder nicht, und in Abhängigkeit davon unter Verwendung der verantwortlichen Methode `createPanelControls` der Platzhalter-Text im Dokument gegen die entsprechenden Steuerelemente in Form von MyXaml-Markup ausgetauscht wird. Weiterhin wird bereits an dieser Stelle eine Instanz der Klasse `PInfofenster` erzeugt, damit später lediglich eine Aktualisierung der Anzeige der detaillierten Informationen zu einem ausgewählten Item notwendig ist. Da das Plug-In die GUI-Definition selbst bereitstellen muss, wird diese als eingebettete Ressource mitgeliefert und erst die modifizierte Version in Form eines neuen Dokumentes auf dem Gerät abgelegt.

Beim Aufruf des Plug-Ins wird schließlich mit Hilfe der Methode `LoadObject` die Klasse `MyPanel` von einem Parser instantiiert und das resultierende Objekt unter dem Namen `contentPanel` zum ClientMainForm hinzugefügt. Die Methode fordert die gleichen Parameter wie die bei der Implementierung des ClientMainForms beschriebene Methode `LoadForm`, ermöglicht es aber im Gegensatz dazu, anstelle eines kompletten Formulares lediglich ein einzelnes Panel o.ä. zu erzeugen. Bei der Instantiierung der Klasse

Alle Rechte vorbehalten

`MyPanel` wird ebenfalls `null` anstelle eines `Parent`-Objektes übergeben, da im Plug-In der aufrufende Client nicht bekannt ist. Aus diesem Grund wird das ClientMainForm erst beim Einbinden des contentPanels in der `LoadESGClient`-Klasse als `Parent`-Objekt des Panels gesetzt.

Die ImageButtons wurden auf die erforderliche Art und Weise in Form von MyXaml-Markup beschrieben und die Modifikation der Klasse ImageButton parallel zur Anpassung des TVZapPlugIns vorgenommen. Auf den ImageButton wird im Anschluss an die Beschreibung des Plug-Ins näher eingegangen.

Die Klassen `MyItemList` und `MyUpdatedItemList` wurden vereinfacht und in Form von Methoden zur Klasse `MyPanel` hinzugefügt. Die ursprünglichen Klassen wurden daraufhin entfernt. Deren Funktionalität wurde dabei nicht verändert, lediglich die von beiden Klassen verwendeten Methoden optimiert und miteinander kombiniert. Anstelle der bisher direkten Erzeugung der Buttons für die Providerliste werden diese in der neuen Version ebenfalls in Form von Markup beschrieben und an die aufrufende Methode in Form einer Zeichenkette übergeben.

Das ursprünglich zum Zeichnen des gesamtes Panels verwendete `Paint`-Event wird in der modifizierten Version ausschließlich dann benötigt, wenn VideoItems verfügbar sind und das Panel zum ersten Mal erstellt wird, da nur dann eine Linie an den rechten Rand der erzeugten Providerliste gezeichnet werden muss, um diese vom InfoPanel abzugrenzen.

Da mittels MyXaml-Beschreibung allem Anschein nach[23] nicht die Möglichkeit besteht, Steuerelemente einer ArrayList hinzuzufügen, müssen die erzeugten ImageButtons daher in einer ArrayList abgelegt werden, sobald das Panel zum ersten Mal vom Client geladen wird und somit den Fokus erhält. Erst durch diese Notlösung ist es möglich, die Buttons wie bisher an das Informationsfenster zu übergeben, welches die Behandlung der Bewegungen des Steuerkreuzes regelt. Aufgrund der Eventbehandlung durch das InfoPanel sowie der Tatsache, dass stets dessen Start-Button fokussiert sein soll, wird, sobald das contentPanel den Fokus erhält, dieser an das InfoPanel weitergegeben. Die Methoden, die im Falle eines Ereignisses aufgerufen werden, konnten unverändert übernommen werden.

Modifikation des ImageButtons

Die Komponente ImageButton wurde auf die im Entwurf angedachte Art und Weise problemlos umgesetzt. Das bedeutet, dass in der modifizierten Version durch das Setzen der

23 Es gibt keinerlei Hinweise auf die Möglichkeit der Beschreibung einer ArrayList in Form von MyXaml-Markup in der Dokumentation. Alle Versuche der Beschreibung blieben erfolglos.

Alle Rechte vorbehalten

Eigenschaft `Focusable` manuell vom Programmierer bestimmt werden kann, ob eine `ImageButton`-Instanz fokussierbar ist oder nicht. Zusätzlich wurde eine Methode `setButtonFocusIndex` hinzugefügt, die es erlaubt, die horizontale und vertikale Position eines ImageButtons in einem 2D-Array, z.B. einer ArrayList, festzulegen. Weiterhin wurde die `OnPaint`-Methode um eine Überprüfung aller vorhandenen bzw. nicht vorhandenen Bilder zur Anzeige der entsprechenden Zustände eines ImageButtons erweitert.

Beim Test des TVZapPlugIns wurden Probleme mit der Eventbehandlung beim Anklicken eines ImageButtons festgestellt. Aus diesem Grund mussten, zusätzlich zu den im Vorfeld analysierten notwendigen Modifikationen an der ImageButton-Klasse, die Verweistypen der Button-spezifischen Events geändert werden. Die nähere Erläuterung dieser Problematik soll im Rahmen der Beschreibung der während der Implementierung aufgetretenen Probleme in Kapitel 6.2. vorgenommen werden.

Modifikation des PInfofensters und der zugehörigen MessageBox

An diesem Punkt der Implementierungsphase bestand bereits eine voll funktionstüchtige, skinningfähige Anwendung. Über den Startbildschirm konnte ein Plug-In ausgewählt und im ClientMainForm angezeigt werden. Es fehlte lediglich die Umsetzung des Informationsfensters zur vollständigen Skinningfähigkeit des ESG-Client.

Im Hinblick auf die Umsetzung wurden zunächst die entsprechenden Klassen in ein separates Projekt ausgelagert sowie die resultierende Klassenbibliothek InfoPanel.dll über einen Verweis vom TVZapPlugIn referenziert. Im Plug-In mussten daraufhin alle Vorkommen des PInfofensters mit dem neuen Namensraum InfoPanel versehen werden, um dieses wie zuvor weiterhin benutzen zu können.

Zunächst wurde die Klasse `PInfofenster` modifiziert. Die GUI-Beschreibung wurde in ein als Ressource eingebettetes MyXaml-Dokument ausgelagert und beim Konstruktoraufruf der Klasse vom Parser anhand der Methode `LoadObject` das entsprechende Panel ohne Angabe eines `Parent`-Objektes erzeugt. Alle definitiv benötigten Steuerelemente werden vom Programmcode erzeugt und im MyXaml-Dokument referenziert sowie die Eigenschaften der Elemente über die zugehörigen Stildefinitionen hinzugefügt. Das Bild für den Medientyp wird zur Laufzeit der Anwendung, abhängig vom übergebenen Item, der entsprechenden PictureBox zugewiesen. Das Bild zur Anzeige des Bezahlstatus wird bereits im MyXaml-Dokument zugewiesen und dynamisch ein- bzw. ausgeblendet. Der Fortschrittsbalken wurde dahingehend verändert, dass nicht bei jedem neu übergebenen Item der alte Balken gelöscht und ein neuer Balken gezeichnet wird, sondern das Steuerelement im

Alle Rechte vorbehalten

Dokument beschrieben und im Programmcode lediglich jeweils der aktuelle Wert gesetzt und die Anzeige aktualisiert wird.

Auch die MessageBox wird in der modifizierten Version nur noch einmalig erzeugt und beim Wechsel des anzuzeigenden Items jeweils die Werte in der Box dem aktuellen Item angepasst. Die Umgestaltung der Klasse `MyMessageBox` erfolgte auf die gleiche Weise wie die des PInfofensters. Im MyXaml-Dokument werden lediglich allen Elementen Stile und Methoden zur Eventbehandlung zugewiesen, die zugehörigen Eigenschaften werden im Stylesheet bzw. dynamisch im Programmcode definiert. Da es sich bei der MessageBox um ein Formular mit benutzerdefinierter Größe handelt, wird zu dessen Instantiierung wieder die Methode `LoadForm` verwendet. Beim Laden des Formulares wird jeweils das oberste Element der enthaltenen ListBox selektiert und fokussiert. Zur Übergabe der aktuellen Werte wurde eine `updateBox`-Methode ergänzt, die als Parameter das aktuelle Item entgegen nimmt und daraufhin die Einträge der ListBox und den Text des Informations-Labels anpasst.

Umsetzung der fehlenden Zapping-Plug-Ins

Aufgrund der Ähnlichkeit aller Zapping-Plug-Ins wurden zuletzt per Copy&Paste sowie minimalen Änderungen im DVB-H AL-Aufruf und an den Item-Typen das AudioZapPlugIn und das ServiceZapPlugIn ergänzt sowie zur Liste der zu ladenden Plug-Ins in der Klasse `LoadESGClient` hinzugefügt, um die Veränderungen der GUI der Anwendung deutlicher sichtbar zu machen.

6.1.2. Phase 2 – Durchführbarkeit des Skinnings

Ziel der zweiten Phase der Implementierung war es, anhand der Erstellung eines leicht modifizierten Skins die Skinningfähigkeit zu testen sowie einen ausgewählten Skin vom Client an alle weiteren Komponenten zu übergeben. In der ersten Phase wurde beim Einbinden des Stylesheets in jeder Komponente der Name von Stylesheet und Skin statisch angegeben und musste somit bisher in jedem MyXaml-Dokument per Hand vom Programmierer verändert werden. Die fest angegebenen Namen wurden zur Erzeugung einer Dynamik in dieser zweiten Phase gegen Platzhalter-Text ausgetauscht. Im ClientMainForm und im StartForm blieben die festen Werte erhalten. In einer Textdatei wurden die alten und neuen Namen des Stylesheets bzw. Skins vermerkt, und in beiden Formularbeschreibungen die alten Namen gesucht und gegen die neuen ersetzt. Allen aufgerufenen Komponenten werden ab sofort jeweils die neuen Namen als Parameter übergeben und der Platzhalter-Text gegen diese ausgetauscht. Dazu wurden zu der Methode getContentPanel der Plug-In-

Alle Rechte vorbehalten

Schnittstelle Parameter zur Übergabe des neuen Stylesheet- und Skinnamens hinzugefügt.

Die erreichte Dynamik ermöglichte im nächsten Schritt die Implementierung des Skinning-Features. Dazu wurde zunächst das in Kapitel 5 entworfene Formular umgesetzt (siehe Abbildung 28). Da dieses v.a. Testzwecken dienen soll, wurde es in der Kürze der Zeit innerhalb einer Methode der Klasse `LoadESGClient` in Form von C#-Code erstellt und ist daher nicht skinningfähig. Das FHTW-Logo auf dem ClientMainForm, ursprünglich eine PictureBox, wird in der neuen Version von einem ButtonEx dargestellt. Beim Klick dieses Buttons wird die Methode `optionsClick` aufgerufen und das Skinning-Formular erzeugt. Dieses listet alle verfügbaren Stylesheets in einer ListBox auf. Im Code wurde festgelegt, dass ausschließlich nach Dateien mit der Endung „*.sts“ im Ordner „Program Files/ESGClient“ gesucht wird.

Abbildung 28: Screenshot des implementierten Skinning-Features

Bei Auswahl eines Stylesheets wird das Event `SelectedIndexChanged` ausgelöst. Zu dessen Behandlung wurde die Methode `stylesheets_SelectedIndexChanged` implementiert, welche das Stylesheet mittels eines `XMLTextReader`-Objektes Tag für Tag nach `MxComponent`-Elementen und dem zugehörigen Attribut `Name` durchsucht und daraufhin alle gefundenen Skins auflistet. Wird ein Skin ausgewählt und die Auswahl über den Ok-Button bestätigt, so wird eine MessageBox mit der Meldung ausgegeben, dass die Anwendung neu gestartet werden muss. Stylesheet und Skin werden vermerkt und beim Schließen des ClientMainForms von einem `StreamWriter`-Objekt in die Textdatei geschrieben. Diese Speicherfunktion wurde nachträglich zur Methode `btnExit_Click` nachträglich hinzugefügt. In die Textdatei werden vier Einträge geschrieben: das zuletzt

verwendete Stylesheet, das neue Stylesheet, der zuletzt verwendete Skin und der neue Skin. Beim nächsten Start der Anwendung wird die Datei erneut ausgelesen und der zuvor ausgewählte Skin verwendet. Dazu werden alle als Strings eingelesenen MyXaml-Dokumente jeweils nach den alten Einträgen durchsucht und die darin befindlichen alten Namen durch die neuen ersetzt. Die Einträge in der Textdatei können auch ohne Verwendung des Skinning-Features geändert werden bevor die Anwendung gestartet wird.

Zum Abschluss der eigentlichen Implementierungsphase wurde ein einfacher Skin erstellt um die Funktionalität testen zu können. Der resultierende modifizierte, weiterhin voll funktionsfähige Startbildschirm ist in der folgenden Abbildung zu sehen. Da nicht die Möglichkeit der Angabe benutzerdefinierter Bildpfade im Stylesheet besteht, wurden zunächst die vorhandenen Bilder in ihren Größen und Positionen verändert. Der Hintergrund des Formulares wurde blau eingefärbt.

Abbildung 29: Beispiel für einen neuen Skin des Startbildschirms

Nachfolgend soll kurz das Ergebnis der Implementierungsphase zusammengefasst und anhand von UML-Diagrammen veranschaulicht werden.

6.1.3. Zusammenfassung

Der Entwurf wurde, abgesehen von minimalen Abänderungen, vollständig umgesetzt. Wie im folgenden Package-Diagramm zu sehen, beschränken sich diese Änderungen in der Klassenstruktur lediglich auf den aussagekräftigeren Namen der ehemaligen Klasse ClientMainForm (siehe 1) sowie die zusätzlich umgesetzten Zapping-Plug-Ins für Audio-Dienste und sonstige Services (siehe 2).

Alle Rechte vorbehalten

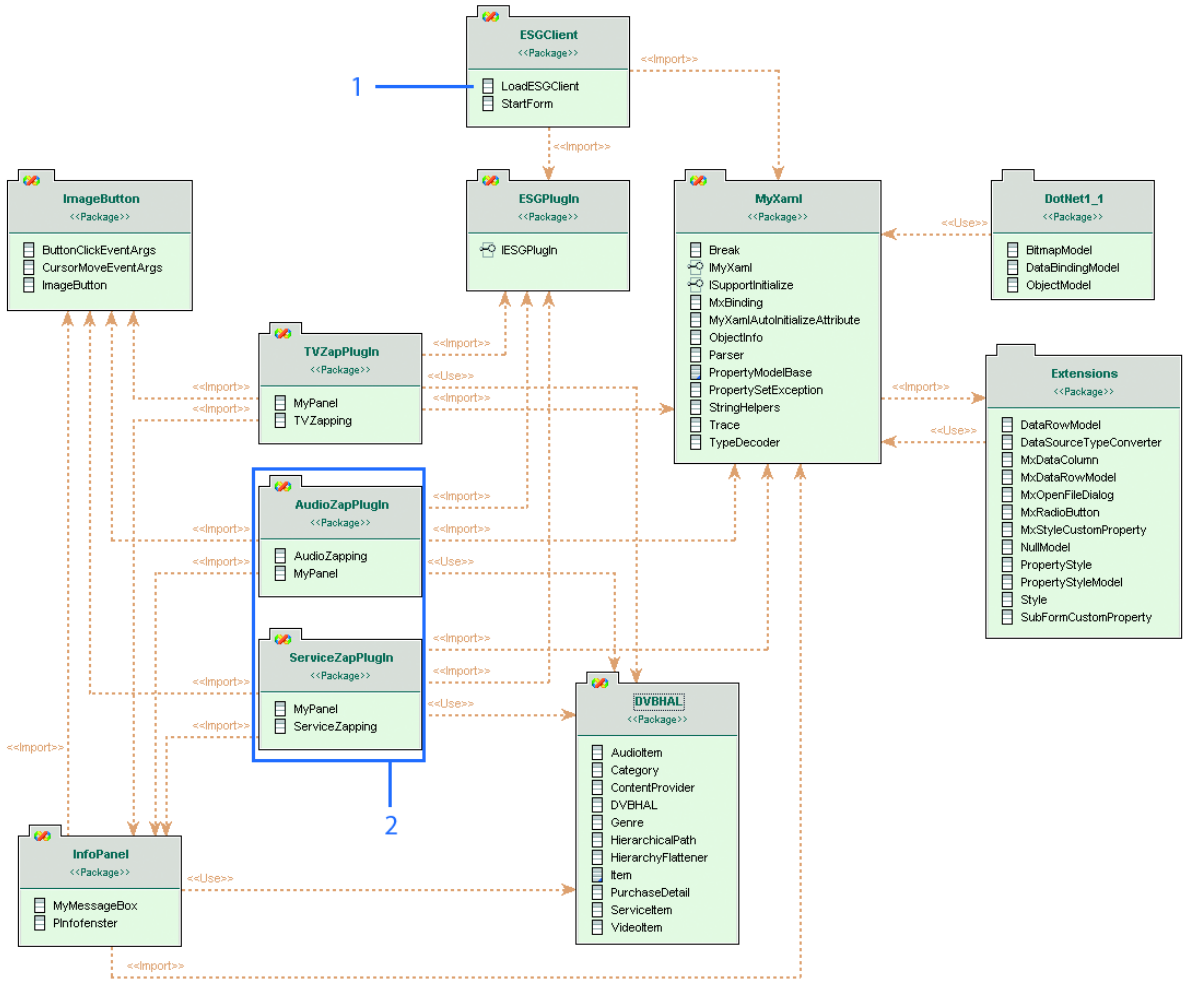

Abbildung 30: Package-Diagramm des resultierenden modifizierten ESG-Client

Die Veränderungen zum ursprünglichen Aufbau der einzelnen Klassen sind anhand der neuen Klassendiagramme zu erkennen, die sich im Anhang B befinden.

Alle benötigten Bilder und Stylesheets sowie die GUI-Beschreibungen für das ClientMainForm und das StartForm müssen im Visual Studio-Projekt `ESGClient` als Inhalt eingebunden und auf dem Endgerät im Ordner „Program Files/ESGClient“ abgelegt werden. Bei der Weitergabe des Projektes an den Emulator geschieht dies automatisch. Die GUI-Beschreibungen der Plug-Ins und des InfoPanels werden von diesen Komponenten als eingebettete Ressourcen selbst bereitgestellt und lediglich die Pfade zu den benötigten Bildern angegeben. Alle zu verwendenden Stylesheets müssen mit der Datei-Endung „*.sts“ abgespeichert sein und Skins in Form von `MxComponent`-Elementen enthalten, damit mit Hilfe des Skinning-Features ein einfaches Branding durchführbar ist.

Während der Implementierungsphase traten zahlreiche Probleme auf. Deren Erläuterung sowie die Benennung der Gründe für deren Auftreten folgen im nächsten Kapitel.

6.2. Problembeschreibungen

Bereits anhand einiger Implementierungsversuche vor der eigentlichen Umsetzung des ESG-Client war zu erkennen, dass die Verwendung der MyXaml-Technologie eine lange Einarbeitungsphase erfordern würde. Der Grund dafür war eine unzureichende Dokumentation des MyXaml-Paketes.

Unzureichende bzw. fehlende Dokumentation

Da es sich bei MyXaml um eine OpenSource-Lösung handelt, ist die Dokumentation in Verbindung mit einem simplen Tutorial in Form eines *Wikis* angelegt, d.h. jeder Benutzer kann, basierend auf seinen eigenen Erfahrungen im Umgang mit der MyXaml-Technologie, Einträge hinzufügen bzw. vorhandene Einträge bearbeiten. Die aufgeführten Beispiele sind daher sehr einfach und auf eine geringe Anzahl begrenzt. Es gibt Ungereimtheiten, z.B. inwiefern Attributwerte in einfachen oder doppelten Anführungszeichen angegeben werden müssen. Weiterhin ist beim Durcharbeiten der Dokumentation unklar, welche Hinweise für welche MyXaml-Versionen gültig sind. Während der Implementierungstests mit der Version für .NET CF konnten einige im Tutorial beschriebenen Codebeispiele nicht umgesetzt werden. Daher schienen alle aufgeführten Beispiele sowie die Dokumentation und das Tutorial für die Desktop-Version optimiert zu sein, welche das volle .NET Framework benutzt.

Unvollständigkeit der MyXamlCF-Version

Eine Webseite[24] im Internet gab Aufschluss darüber, dass es sich bei MyXamlCF um eine Adaption des ursprünglich für das volle .NET Framework entwickelten MyXaml-Paketes durch einen anderen Autor handelte und erläuterte einige Unterschiede in der Programmierung. Weiterhin wurden einige fehlende Funktionen genannt sowie der Hinweis gegeben, dass Teile der adaptierten Version noch immer weit von einer vollständigen Funktionsfähigkeit entfernt sind. Zusätzlich wurde ein Beispiel bereitgestellt, welches speziell zur Ausführung auf einem Pocket PC entwickelt wurde. Diese exemplarische Implementierung beseitigte einige Unklarheiten, wie z.B. die Verwendung von code-behind-Code, die zuvor lediglich in der Theorie im Tutorial angerissen, jedoch nirgendwo zu Demonstrationszwecken umgesetzt worden ist. Mit Hilfe dieses Beispiels konnten letztendlich erste Erfolge im Rahmen der Implementierungsversuche erzielt werden.

Die unzureichende Dokumentation des MyXaml-Paketes sowie insbesondere das Fehlen einer Dokumentation der MyXamlCF-Version sind auch die Gründe für alle folgenden

24 Siehe http://www.codeproject.com/netcf/UsingMyXamlCF.asp

Alle Rechte vorbehalten

Schwierigkeiten bei der Umsetzung des skinningfähigen ESG-Client. Die Implementierungsphase wurde somit ebenfalls, wie auch zuvor bereits die Einarbeitungsphase, unnötig ausgedehnt.

Fehlende Unterstützung zur Verwendung von Bildern

Zu den genannten Schwierigkeiten gehörte u.a. die Arbeit mit Bildern. Die im Tutorial gezeigten Beispiele funktionierten auf dem PDA nicht. Es war nicht möglich, mittels Markup Bilder für die Steuerelemente zu definieren und diese den Elementen bei deren Instantiierung automatisch durch den Parser zuweisen zu lassen. Es wurde weder ein Bild angezeigt noch gab es eine Fehlermeldung. Daher wurde die einzige Quelle zu Rate gezogen, die bei Problemen verwendet werden konnte. Dabei handelte es sich um ein Forum[25] auf der MyXaml-Webseite, welches speziell für die Arbeit mit der CF-Version eingerichtet wurde. Dieses war jedoch wenig besucht. Daher liegt die Annahme nahe, dass es kaum erfahrene Benutzer im Umgang mit MyXamlCF gibt. Eine Anfrage im Forum zum erwähnten Problem ergab die Antwort von Bert Bierman, dem Autor der MyXamlCF-Version und Leiter des Forums, dass diese Funktion in der CF-Version gar nicht implementiert ist und dieser Hinweis bisher von ihm selbst nirgendwo vermerkt und das Problem auch noch nicht entdeckt wurde. Es gab jedoch einen hilfreichen Lösungsvorschlag und daraufhin eine neue CF-Version, die es ermöglichte, Bilder auf die im Tutorial beschriebene Art und Weise mittels MyXaml-Markup zu laden und den betreffenden Steuerelementen zuzuweisen (Beispiel siehe Abschnitt 5.1.2. Abbildung 23).

Unvollständiger und fehlerhafter TypeDecoder

Ein weiteres Problem stellte die Verwendung einiger Element-spezifischer Eigenschaften dar. Beim Erzeugen der beschriebenen GUI-Elemente fiel auf, dass alle Buttons umrandet waren, obwohl die Eigenschaft `BorderStyle` auf den Wert `None` gesetzt war. Nach langer Suche im Programmcode des MyXaml-Parsers wurde klar, dass dies eine der fehlenden Funktionen im unvollständigen TypeDecoder war. Die Dekodierung der entsprechenden Eigenschaft wurde hinzugefügt, das MyXaml-Projekt neu kompiliert und die Ränder daraufhin nicht mehr angezeigt. Im Laufe der Implementierungsphase trat es häufiger auf, dass einige Eigenschaften vom Parser ignoriert wurden. Da die Lösung jedoch zu dem Zeitpunkt bereits bekannt war, konnte das jeweilige Problem dementsprechend schnell behoben werden. Weiterhin ist aufgefallen, dass alle Farben, die im Markup direkt über RGB-Werte angegeben wurden, nicht erkannt und statt dessen die Farbe schwarz verwendet wurde. Die Dekodierung des Wertes für das `Color`-Attribut war bereits im TypeDecoder implementiert, wie sich heraus stellte war

25 Siehe http://66.139.77.10/forum/?f=26

Alle Rechte vorbehalten

diese Implementierung jedoch fehlerhaft. Die vorhandene Methode wurde korrigiert und die RGB-Werte konnten zur Darstellung beliebiger Farben verwendet werden.

Fehlende Unterstützung zur Verwendung von Bildern in Stylesheets

Bei dem Versuch der Auslagerung der GUI-spezifischen Eigenschaften der Steuerelemente stellte sich heraus, dass die Bildanzeige erneut nicht funktionierte. Der Grund dafür schien die Klasse `StyleModel` des MyXaml-Parsers zu sein, die eine Definition eines Bitmap-Elementes nicht als Stil-Eigenschaft erkannt hat. Aufgrund des begrenzten Zeitfensters wurde diese Problematik jedoch zunächst an das Ende der Implementierungsphase verschoben. Letztendlich reichte die Zeit zur Beschäftigung mit dem zugehörigen Code nicht aus, um das Problem zu lösen. Da auch Bert Bierman keinen Lösungsvorschlag wusste, konnte die benutzerdefinierte Austauschbarkeit von Bildern anhand von Pfadangaben im Stylesheet im Rahmen dieser Arbeit nicht realisiert werden. Daher werden Bilder im gegenwärtigen modifizierten ESG-Client direkt im MyXaml-Dokument oder zum Teil im Programmcode über Pfadangaben zugewiesen.

Unzureichende Generizität des Parsers

Im Anschluss an die erwähnte Problematik der Bildanzeige ergab sich das Problem, dass beim Anklicken eines ImageButtons eine Fehlermeldung ausgegeben und das Programm daraufhin beendet wurde. Beim Debuggen der Anwendung konnte lediglich herausgefunden werden an welcher Stelle der Fehler auftrat, der Grund ließ sich jedoch auch nach längerer Suche nicht feststellen. An der ermittelten Stelle wurde lediglich ein Eventhandler angebunden wurde. Wie sich in Kommunikation mit dem Entwickler der MyXamlCF-Version herausstellte, war die Instantiierung des Eventhandlers im MyXaml-Parser nicht generisch genug, um den `ButtonClickedEventhandler` der ImageButton-Komponente zu verwenden. In der Kürze der Zeit konnte von Seiten des MyXamlCF-Autors nur eine vorübergehende Notlösung implementiert werden. Es wurde wieder eine neue Version von MyXamlCF erzeugt.

Mit Hilfe dieser zweiten neuen Version und einer minimalen Änderung im Programmcode des ImageButtons konnte dieser problemlos vom Parser instantiiert und das zugehörige `ButtonClicked`-Event ausgeführt werden. Zuvor wurde beim Drücken des ImageButtons das `ButtonClicked`-Event durch den Button-spezifischen Eventhandler `ButtonClickedEventhandler` aufgerufen. Da in MyXaml beim Instantiieren der Klassen hinter den Tags jeweils der `System.EventHandler`, welchen .NET bereitstellt, verwendet wird, gab es Konflikte mit dem Button-spezifischen Eventhandler. Somit mussten alle in der ImageButton Klasse erstellten Eventhandler entfernt und für alle Events der

Alle Rechte vorbehalten

Standard-Eventhandler von .NET als Delegate[26] (engl. für Verweistyp) verwendet werden.

Fehlende Erläuterung des Parsers

Ein weiteres Problem, das auf die unzureichende Dokumentation des MyXaml-Paketes zurückzuführen ist, trat bei der Erstellung der Plug-Ins auf. Es sollte wie im ursprünglichen ESG-Client lediglich ein `Panel`-Objekt aus einer MyXaml-Beschreibung erzeugt werden. Alle Beispiele in der Dokumentation verwendeten jedoch Formulare. Es gab keinen Hinweis darauf, ob es möglich war einzelne Objekte zu instantiieren. Am `Parser`-Objekt konnten verschiedene Methoden ausgeführt werden, für die es allerdings keinerlei Erklärung zu deren Bedeutung und Funktionsweise gab.

Die Methode `LoadObject` schien anstelle der Methode `LoadForm` in Frage zu kommen, der Aufbau des benötigten Dokumentes war jedoch unklar. Durch Ausprobieren stellte sich daraufhin heraus, dass es ausreichend ist, ein typisches MyXaml-Dokument zu erstellen und direkt nach der XML-Deklaration und dem MyXaml-Element anstelle eines Formulares das benötigte `Panel`-Element einzufügen. Im Gegensatz zur Instantiierung eines Formulares mittels der Methode `LoadForm` musste jedoch das verwendete Stylesheet innerhalb des `Controls`-Elementes eingebunden werden. Der Grund dafür ist weiterhin unklar. Im Formular kann es direkt nach dem Start-Tag des Formulares eingebunden werden und muss nicht innerhalb des `Controls`-Elementes aufgeführt sein.

Fehlende Hinweise zur Deklaration von Methoden

Weiterhin wurde an keiner Stelle der Dokumentation der Hinweis gegeben, dass alle Methoden, die beim Eintreten bestimmter Ereignisse aufgerufen werden, öffentlich sein müssen. Diese Erkenntnis wurde anhand zahlreicher Tests während der Implementierungsphase gewonnen.

Die in diesem Kapitel erläuterten Probleme führten dazu, dass aufgrund von Zeitmangel einige der geforderten Funktionalitäten nicht optimal bzw. gar nicht implementiert werden konnten. Auch das Testen der realisierten Funktionen konnte daher nur in geringem Umfang durchgeführt werden. Die resultierenden Testergebnisse werden im nachfolgenden Abschnitt diskutiert.

Alle Rechte vorbehalten

26 Siehe http://msdn.microsoft.com/library/deu/default.asp?url=/library/DEU/csref/html/vcrefTheDelegateType.asp

6.3. Testergebnisse

Neben den kontinuierlich durchgeführten Tests während der Implementierungsphase wurde zusätzlich ein Test mit einer außenstehenden Testperson vorgenommen. Ziel dieses Versuches war es, zu ermitteln, inwieweit die grafische Benutzungsschnittstelle der modifizierten Anwendung in ihrer Gestaltung benutzerdefiniert anpassbar ist und ob das Ziel erreicht wurde, dass das Branding des ESG-Client von einer Person mit grundlegenden Programmierkenntnissen, in diesem Fall einem Programmierer, Screendesigner oder sonstigen Angestellten der TSI, durchführbar ist. Das Skinning-Feature war zu diesem Zeitpunkt noch nicht fertig gestellt. Ein PDA oder die Entwicklungsumgebung Visual Studio .NET 2003 mit einem Pocket PC-Emulator standen ebenfalls nicht für die Testperson zur Verfügung.

Bei der Testperson handelt es sich um einen 24-jährigen, männlichen Studenten der Internationalen Medieninformatik. Die ursprüngliche Anwendung war der Testperson durch eine einmalige Vorführung bekannt. Es wurde die Aufgabe gestellt, ein neues Stylesheet-Dokument mit einem Skin zu erzeugen. Als Hilfestellung standen das vom ESG-Client verwendete Stylesheet-Dokument `Stylesheet.sts` mit dem darin definierten `StandardSkin` sowie eine kurze Anleitung zur Verfügung. Diese enthielt neben Hinweisen zur Vorgehensweise u.a. Screenshots der einzelnen Ansichten des ESG-Clients sowie eine Auflistung aller verfügbaren Eigenschaften und zugehörigen Werte der in der Anwendung verwendeten Steuerelemente. Diese Skinning-Anleitung ist als PDF-Dokument auf der beigefügten CD-ROM enthalten.

Von der Testperson wurde das MyXaml-Dokument `TestUserStylesheet.sts` mit dem darin definierten `TestUserSkin` erstellt. Mit Hilfe des Skinning-Features konnten zu einem späteren Zeitpunkt Stylesheet und Skin problemlos der Anwendung zugewiesen und damit sowohl die Skinningfähigkeit des ESG-Client als auch die Fähigkeit außenstehender Personen zur Erstellung von Stylesheet-Dokumenten und Skins getestet werden. Das Testergebnis für den Startscreen ist in Abbildung 31 zu sehen. Das Ziel der Testperson war es dabei, die Titelzeile, den Schließen-Button und das FHTW-Logo in ihrer Höhe zu vergrößern, den Hintergrund des Formulares blau einzufärben sowie die Plug-In-Buttons zu verkleinern.

Alle Rechte vorbehalten

Abbildung 31: Screenshot des Startbildschirms mit TestUserSkin (fehlerhaft)

Beim Vergleich mit dem zugehörigen MyXaml-Dokument fiel auf, dass die Größe der Titelzeile anscheinend unverändert geblieben ist, während die gleiche Höhenänderung beim Schließen-Button übernommen wurde. Die einzelnen Buttons zur Repräsentation der Plug-Ins besitzen die im Skin angegebene Höhe, behalten jedoch weiterhin die im Programmcode berechnete Position unabhängig von dem neuen Wert der Button-Höhe. Die vom Tester gewählte Hintergrundfarbe des StartForms wurde problemlos übernommen. Das FHTW-Logo ist nicht mehr zu sehen, da die Titelzeile aufgrund der größeren Höhe diesen Bereich verdeckt. Um das zu vermeiden hätte dem Logo eine andere Position zugewiesen werden müssen.

Die Auswirkungen des `TestUserSkin` auf das ClientMainForm, das TVZapPlugIn und das InfoPanel sind im nächsten Screenshot zu sehen. In diesem Fall sollte die Hintergrundfarbe des Formulares und des InfoPanels blau sein, die Höhe der Titelzeile und des Schließen-Buttons vergrößert sowie das TVZapPlugIn aufgrund der Höhenänderung weiter nach unten verschoben werden. Die Buttons zur Navigation in der Anbieterliste des Plug-Ins sollten in ihrer Breite verkürzt, der Hintergrund der Liste gelb sowie die Schrift und die Rahmen um die Einträge blau gefärbt werden. Der Titel des InfoPanels sollte größer werden. Weiterhin war die Einblendung der Prozentzahl des Fortschrittsbalken erwünscht.

Alle Rechte vor...

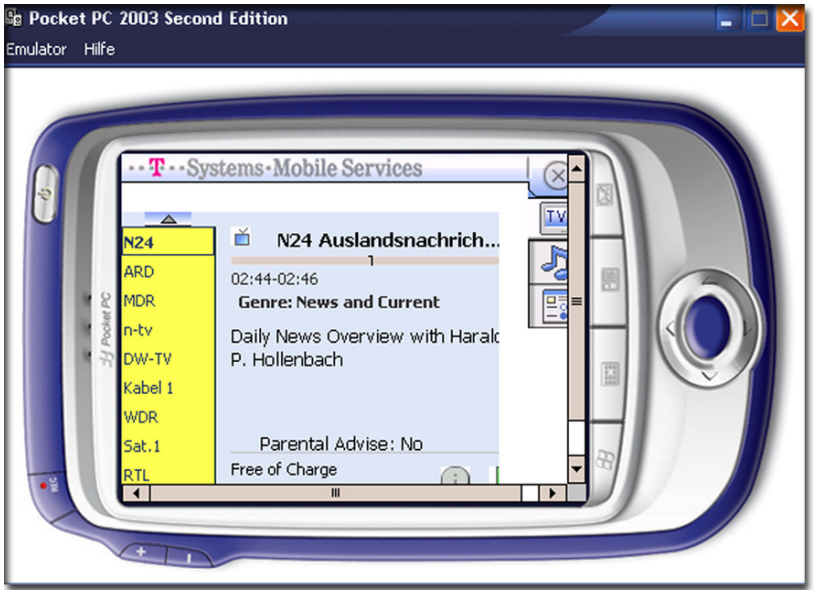

Abbildung 32: ClientMainForm, TVZapPlugIn und InfoPanel mit TestUserSkin (fehlerhaft)

Wie in der vorhergehenden Abbildung ist erneut zu erkennen, dass die Änderung in der Höhe der Titelzeile nicht übernommen wurde. Beim genauen Vergleich mit den im Skin angegebenen Werten fiel jedoch auf, dass auch die Hintergrundfarbe und Änderung der Größe des Schließen-Buttons nicht angezeigt werden. Daher scheint ein Fehler bei der Weitergabe des aktuellen Skins an das ClientMainForm vorzuliegen, der während der Implementierungsphase und zwischenzeitlichen Tests soweit unentdeckt blieb. Die Rollbalken am rechten und unteren Rand lassen vermuten, dass die Veränderung von Positionen und Größen von Steuerelementen problematisch ist, wenn die Auswirkungen dieser Änderungen nicht parallel zur Erstellung des Skins sichtbar sind.

Aufgrund dieser Testergebnisse wurden sowohl der `TestUserSkin` als auch die modifizierte Anwendung erneut geprüft, um die aufgetretenen Fehler finden und beseitigen zu können. So wurde u.a. festgestellt, dass eine Größenänderung der Titelzeile nicht möglich war, weil vom Tester die Eigenschaft `SizeMode` für das Steuerelement `PictureBox` nicht auf den Wert `StretchImage` gesetzt wurde. Diese Eigenschaft legt fest, dass das Bild unabhängig von dessen eigener Größe auf die Größe des Steuerelementes gesetzt wird. Als auch nach Hinzufügen dieser Eigenschaft das Bild nicht vergrößert dargestellt wurde, lag die Vermutung nahe, dass es sich erneut um eine Eigenschaft handelt, die nicht vom MyXaml-Parser dekodiert werden konnte. Somit wurde die Dekodierung in der Klasse `TypeDecoder` ergänzt und das Bild letztendlich an die Größe der PictureBox angepasst. Der tatsächlich von der Testperson gewünschte Startbildschirm sieht nach der Korrektur im Programm folgendermaßen aus:

Alle Rechte vorbehalten

Abbildung 33: Screenshot des Startbildschirms mit TestUserSkin

Anschließend wurde die Zuweisung des jeweils aktuellen Stylesheets und Skins an das ClientMainForm in der Klasse `LoadESGClient` geprüft. Dabei ist aufgefallen, dass noch vor der Zuweisung bereits die alten Werte durch die neuen ersetzt werden.

Die Namen des zuletzt verwendeten und beim nächsten Start der Anwendung zu verwendenden Stylesheets und Skins werden, wie bei der Implementierung beschrieben, jeweils in eine Textdatei gespeichert wenn die Anwendung beendet wird. Wurden zuvor mittels des Skinning-Features Stylesheet und Skin neu gewählt, so werden diese und die zu dem Zeitpunkt verwendeten in der Textdatei vermerkt. Beim nächsten Start des ESG-Client werden alle Werte ausgelesen, die MyXaml-Dokumente nach den alten Namen durchsucht und gegen die neuen ersetzt sowie im Anschluss daran die neuen Namen als die aktuell verwendeten vermerkt.

Da in der Anwendung die neuen Namen zu frühzeitig als die alten bzw. aktuell verwendeten Namen vermerkt wurden, konnten somit bei der Suche im MyXaml-Dokument nach den alten Werten der vorigen Stylesheets und Skins deren Namen nicht gefunden und durch die neuen ersetzt werden, da bereits nach den neuen Werten anstelle der alten gesucht wurde. Daher wurden die Namen des zu verwendenden Stylesheets und Skins nicht ausgetauscht und weiterhin die vor dem Neustart verwendeten beibehalten. Die Aktualisierung des zuletzt verwendeten Stylesheets und Skins mit den neuen Werten wurde daraufhin später im Programmablauf vorgenommen und die neu zugewiesenen Werte vom ClientMainForm wie gewünscht übernommen (siehe Abbildung 34).

Alle Rechte vor...

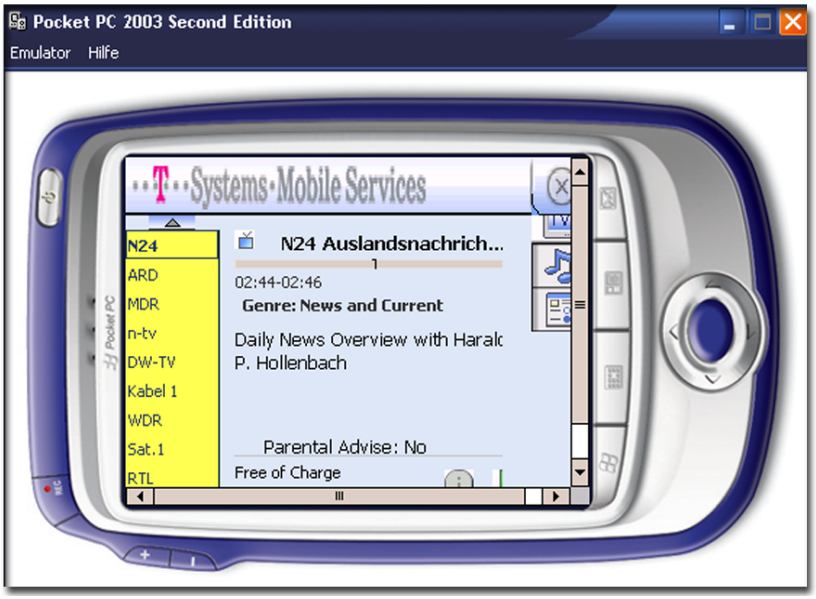

Abbildung 34: ClientMainForm, TVZapPlugIn und InfoPanel mit TestUserSkin

An der genannten Problematik der Bestimmung von Größen und Positionen der Steuerelemente ist erkennbar, dass zum einen die Programmierung stellenweise nicht dynamisch genug ist, und sich zum anderen die Skin-Definition ohne eine gleichzeitige Ansichtsmöglichkeit in diesem Zusammenhang als zu kompliziert erweist.

Das Problem innerhalb der Programmierung besteht darin, dass in der ursprünglichen Version alle Größen festgelegt und anhand dieser konstanten Größen u.a die mögliche Anzahl dynamisch erzeugter Buttons sowie deren Position berechnet wurde. In einer verbesserten Version des skinningfähigen ESG-Client müssten daher alle Berechnungen auf Basis der im Stylesheet angegebenen Werte erfolgen. Weiterhin hat die Testperson detailliertere Erklärungen zur Bedeutung der einzelnen Eigenschaften der Steuerelemente in der Skinning-Anleitung vermisst sowie die Reihenfolge der Anordnung und unzureichende Abgrenzungen zwischen den Stildefinitionen bemängelt. Diese Kritik sollte ebenso in einer neuen Version der Anwendung und der Skinning-Anleitung beachtet werden.

Die während der Testphase vorgenommenen Veränderungen haben keinen weiteren Einfluss auf die zuvor dargestellte Klassenstruktur des modifizierten ESG-Client genommen. Daher kann an dieser Stelle auf eine neue Darstellung verzichtet werden. Das Ergebnis der in den vorhergehenden Abschnitten beschriebenen Implementierung soll im folgenden Kapitel mit den gestellten Anforderungen verglichen und ausgewertet werden.

Alle Rechte vor

7. Ergebnisse

Eine genaue Betrachtung der im Rahmen der vorliegenden Diplomarbeit umgesetzten Neugestaltung des prototypischen ESG-Client zeigt, dass die gesetzte Zielstellung weitestgehend erreicht wurde. Basierend auf einer ausführlichen Analyse der bestehenden Implementierung konnte unter Verwendung der neuartigen Technologie MyXaml eine Anwendung geschaffen werden, deren grafische Benutzungsschnittstelle schnell und einfach, frei von jeglichem Programmieraufwand individuell an verschiedene Designs angepasst werden kann, ohne dass eine Veränderung der unterliegenden Applikationslogik erforderlich ist und die wie gefordert auf der Plattform Pocket PC 2003 SE lauffähig ist.

Auf die erzielten Resultate aber auch die fehlende Funktionalität soll, im Vergleich mit den an die modifizierte Anwendung gestellten Anforderungen sowie der konkreten Zieldefinition, nachfolgend im Einzelnen noch einmal genauer eingegangen werden:

Umsetzung der Skinningfähigkeit

Im Rahmen dieser Arbeit konnte die zwingend erforderliche Trennung der GUI von der unterliegenden Applikationslogik insofern realisiert werden, dass sämtliche Formulare und Panels sowie die zugehörigen Interaktionselemente in Form von XML-basiertem Markup beschrieben und als MyXaml-Dokumente gespeichert werden. In der Applikationslogik wiederum werden mit Hilfe von C#-Anweisungen Instanzen des MyXaml-Parsers erzeugt, welche die Verarbeitung dieser Dokumente vornehmen, indem die unterliegenden Klassen instantiiert, Eigenschaften initialisiert und Eventhandler angebunden werden.

Jedem einzelnen Steuerelement wird dabei mittels der Style-Eigenschaft von MyXaml eine individuelle Stildefinition zugewiesen. Diese Stildefinitionen umfassen, abgesehen von essentiellen Bildzuweisungen, alle zur Darstellung der Elemente auf dem Bildschirm notwendigen Eigenschaften. Dazu zählen ebenfalls, wenn auch eingeschränkt, die als optional gekennzeichneten Eigenschaften Größe und Position. Durch eine Veränderung der Eigenschaften der entsprechenden Stildefinitionen kann somit die geforderte benutzerdefinierte Interfacegestaltung der Anwendung erfolgen.

Die Modifikation des ImageButtons, die als zwingend erforderlich eingestuft wurde, um die Komponente weiterhin auf die in der ursprünglichen Version verwendete Art und Weise benutzen zu können, wurde wie geplant realisiert. Die Beschreibung eines ImageButtons mittels Textauszeichnung sowie die automatische Instantiierung der zugehörigen Klasse durch den MyXaml-Parser ist daher problemlos möglich.

Alle Rechte vorbehalten

Durchführbarkeit des Skinnings

Die geforderte schnelle und einfache Anpassbarkeit der GUI konnte insofern umgesetzt werden, dass mittels des implementierten Skinning-Features alle verfügbaren Stylesheets und darin definierten Skins schnell angezeigt und ganz einfach per Knopfdruck ausgewählt werden können. Die Erstellung der einzelnen Skins ist jedoch, wie in den Anforderungen bereits beschrieben, etwas zeitaufwändiger und komplizierter, da die einzelnen Stile der Elemente direkt in Form von MyXaml-Markup definiert werden müssen. Es ist daher prinzipiell möglich, ein einfaches Branding von einem TSI-Mitarbeiter mit grundlegenden Programmierkenntnissen durchführen zu lassen. Zukünftig könnte jedoch eine GUI zur Erstellung von Skins hilfreich sein, welche die Auswirkungen der Veränderung von Eigenschaften zur Darstellung eines Elementes direkt anzeigt, damit das Branding zum einen auch von Angestellten ohne jegliche Programmierkenntnisse und zum anderen wesentlich schneller und nahezu fehlerfrei durchgeführt werden kann.

Die bei der momentanen Vorgehensweise zur Skin-Definition als Hilfestellung benötigte Skinning-Anleitung konnte ebenfalls verfasst werden und ist auf der beigelegten CD zu finden. Diese Anleitung gibt Hinweise zu Aufbau und Definition von Stylesheets und Skins sowie einen Überblick über alle verwendbaren Eigenschaften der Steuerelemente und deren mögliche Werte. Die Demonstration der Skinningfähigkeit konnte während der Testphase problemlos erfolgen und Fehler in der Implementierung dadurch aufgedeckt und beseitigt werden.

Fehlende Funktionalität

Wie im vorhergehenden Abschnitt erwähnt, ist die schnelle Anpassbarkeit mittels des Skinning-Features realisiert worden. Während der parallel zur Implementierungsphase durchgeführten Tests wie auch beim anschließenden Testen der resultierenden Anwendung wurde festgestellt, dass die Anwendung wesentlich langsamer startet als es bei dem ursprünglichen ESG-Client der Fall war. Es kann jedoch nicht gesagt werden, ob dies ausschließlich auf dem Emulator der Fall ist oder ob die modifizierte Anwendung auch auf einem richtigen Pocket PC unter wesentlichen Performanzeinbußen zu leiden hat, da kein Testgerät zur Verfügung stand.

Weiterhin wurde zuvor bereits eine Einschränkung in der realisierten Skinning-fähigkeit hinsichtlich der Anpassbarkeit aller GUI-Elemente gemacht, welche die optionale Veränderung von Größen und Positionen als auch die essentielle Austauschbarkeit von Bildern betrifft. Größen und Positionen können generell über ein Stylesheet verändert werden,

Alle Rechte vorbehalten

in einigen Fällen werden diese Werte jedoch nicht beachtet, da die Darstellung der Elemente im Programmcode, basierend auf der originalen Implementierung, anhand festgelegter Werte für Größen und Positionen automatisch erfolgt. Dies trifft u.a. für die Erstellung der Tableiste und der Anbieterlisten in den Zapping-Plug-Ins zu.

Eine benutzerdefinierte Austauschbarkeit von Bildern konnte nicht optimal realisiert werden. Ohne Veränderungen im Programmcode vorzunehmen ist es ausschließlich möglich, die in dem fest angegebenen Ordner „Program Files/ESGClient“ abgelegten Bilder gegen namensgleiche Bilder auszutauschen. Eine Angabe von Pfaden zu den zu verwendenden Bildern kann nur in den MyXaml-Dokumenten der jeweiligen GUI-Beschreibung bzw. zum Teil im Programmcode erfolgen. Eine Angabe beliebiger Bildpfade im Stylesheet ist nicht möglich, da der MyXaml-Parser dies bisher nicht unterstützt und es in der Kürze der Zeit nicht möglich war, die notwendigen Modifikationen im Programmcode des Parsers vorzunehmen.

Da die Konzentration auf der Umsetzung der geforderten Funktionalität lag wurde keinerlei Fehlerbehandlung in der neugestalteten Version des ESG-Client implementiert. So ist dies u.a. zu erkennen, wenn in der Textdatei keine oder falsche Namen für die zu ladenden Stylesheets und/oder Skins angegeben sind. Ist dies der Fall, startet die Anwendung nicht und es wird eine Fehlermeldung von Windows CE ausgegeben, die lediglich auf eine „Exception“ hinweist, jedoch den genauen Grund nicht kennt. Auf weitere bekannte Probleme wird im Benutzerhandbuch hingewiesen.

Bei genauerer Betrachtung der umgesetzten und fehlenden Funktionalität lässt sich feststellen, dass die Erwartungen an die resultierende modifizierte Anwendung im Großen und Ganzen, abgesehen von einigen Abstrichen, erfüllt wurden. Der ESG-Client ist weitestgehend benutzerdefiniert anpassbar in seiner Interfacegestaltung und ein einfaches Skinning kann problemlos durchgeführt werden. Als problematisch zu betrachten ist jedoch die resultierende Performanz. Das Starten der Anwendung sowie jeder Wechsel des angezeigten Plug-Ins benötigen ca. eine halbe Minute. Als Testsystem diente in diesem Fall ein Acer TravelMate 800 mit dem Betriebssystem Windows XP Professional, einem 1,3 GHz Intel Centrino-Prozessor und 512 MB Arbeitsspeicher. Getestet wurde die Anwendung im Pocket PC 2003 SE-Emulator. Da es sich beim ESG-Client um eine Anwendung handelt, die auf dem Pocket PC unterwegs in Bus und Bahn sowie in kurzen Pausen benutzt werden soll um u.a. schnell an bestimmte Informationen zu gelangen, ist es für Endkunden unzumutbar eine halbe Minute zu warten bis sich das Programm geöffnet hat, auch wenn alle Aktualisierungen der Anzeige

Alle Rechte vorbehalten

daraufhin zum Teil sogar schneller als zuvor erfolgen. Die entwickelte Lösung bietet sich daher zwar für das Branding durch die TSI an und erfüllt somit die Aufgabenstellung, ist jedoch im Hinblick auf eine alltägliche Verwendung durch Endbenutzer absolut ungeeignet.

Der Grund für die angeführte Problematik kann sowohl die zur Modifikation verwendete Technologie MyXaml als auch das unterliegende Plug-In-Konzept sein. Ebenso kommen eine ungünstig umgesetzte Implementierung und der u.U. ohnehin bereits langsame Pocket PC-Emulator als Gründe für die schlechte Performanz in Frage.

Da es sich bei MyXaml um eine neue Entwicklung handelt, die zudem eine OpenSource-Implementierung einiger weniger Personen anstelle eines großen professionellen Teams darstellt, die Implementierung noch weitgehend unvollständig und MyXamlCF lediglich eine Adaption dieser Technologie ist, ist es durchaus denkbar, dass diese Technologie einer wesentlichen Verbesserung bedarf um eine bessere Performanz der entwickelten Software erzielen zu können. Das Auslesen der Dokumente, die spätere Zuordnung und Instantiierung der entsprechenden Klassen sowie die Initialisierung der erzeugten Objekte mittels ihrer Eigenschaften und die Anbindung der Events scheinen ein zu langwieriger Prozess zu sein, als dass der Parser diese Aufgaben in Rekordzeit lösen könnte.

Hinzu kommt das Plug-In-Konzept, welches bei der Modifikation erhalten bleiben sollte um weiterhin die Möglichkeit einer Austauschbarkeit der Komponenten zu bieten. Zur Erzeugung jeder einzelnen Komponente wird zunächst ein eigenes Parser-Objekt erzeugt, welches die jeweilige GUI-Beschreibung verarbeitet und an die aufrufende Komponente übergibt. Möglicherweise hätte das Plug-In-Konzept verworfen oder aber auf andere Weise implementiert werden müssen, um einen schnelleren Ablauf der modifizierten Anwendung zu erzielen. Mittels umfangreicher Tests könnten verschiedene Lösungen entwickelt und umgesetzt werden um das Problem zu finden. So wäre es u.a. eine Idee, alle GUI-Beschreibungen in einem gemeinsamen MyXaml-Dokument zu speichern und dieses lediglich von einem Parser-Objekt verarbeiten zu lassen. Ebenso gut ist es möglich, dass die Anwendung auf einem richtigen Pocket PC wesentlich schneller abläuft und keinerlei Änderungen notwendig sind. Bereits im zugrunde liegenden DVB-H-Projekt bestand während der Implementierungsphase aufgrund von parallelen Tests mit dem Emulator der Verdacht, dass die Anwendung zu langsam und daher unbrauchbar wäre. Beim Test auf einem richtigen Pocket PC erwies sich dieser Verdacht als falsch. Daher müsste ein Pocket PC beschafft und die modifizierte Anwendung zunächst einmal darauf getestet werden.

Alle Rechte vorbehalten

Die notwendigen Tests als auch die Implementierung einer Fehlerbehandlung konnten aufgrund der unzureichenden Dokumentation und der Unvollständigkeit des MyXaml-Parsers nicht im Rahmen dieser Arbeit durchgeführt werden. Die erforderliche parallele Erweiterung der Parser-Funktionalität wirkte zum Teil behindernd im Vorankommen mit der eigenen Implementierung, war jedoch andererseits sehr von Vorteil, da einige Funktionalitäten u.U. nicht wie geplant hätten umgesetzt werden können ohne diese Erweiterungsmöglichkeit. Daher ist die Entscheidung für die verwendete OpenSource-Technologie MyXaml positiv zu bewerten.

Abschließend lässt sich jedoch sagen, dass mit der gegenwärtig verfügbaren MyXamlCF-Version sowie der Beibehaltung und aktuellen Umsetzung des Plug-In-Konzeptes zwar ein die Anforderungen erfüllendes funktionsfähiges, jedoch kein optimales Ergebnis erzielt werden konnte, sofern eine ähnlich geringe Geschwindigkeit des Programmablaufes auf einem richtigen Pocket PC anzunehmen ist. Im Rahmen einer Weiterführung der vorliegenden Arbeit könnten die fehlenden Funktionen voraussichtlich problemlos hinzugefügt werden, sollten Tests jedoch ergeben, dass keine bessere Performanz auf einem richtigen Pocket PC erzielt werden kann, so sollte entweder über die Verwendung einer verbesserten Version von MyXamlCF bzw. eines anderen Lösungsansatzes nachgedacht oder aber das Plug-In-Konzept auf eine optimalere Art und Weise implementiert bzw. verworfen werden.

Im nachfolgenden und zugleich letzten Kapitel sollen die wichtigsten Punkte der vorliegenden Arbeit noch einmal kurz zusammengefasst sowie Ideen für eine mögliche Weiterführung genannt werden.

Alle Rechte vorbehalten

8. Zusammenfassung

Gegenstand der vorliegenden Diplomarbeit ist die Problematik der Trennung der grafischen Benutzungsschnittstelle und der unterliegenden Applikationslogik. Im Rahmen dieser Arbeit bestand die konkrete Anforderung, eine prototypische Implementierung eines ESG-Client für DVB-H individuell anpassbar in der Gestaltung der grafischen Benutzungsschnittstelle zu machen, um ein Branding der Anwendung für verschiedene Firmen entsprechend deren Corporate Design zu ermöglichen. Das Ziel bestand darin, die notwendige Skinningfähigkeit für einige ausgewählte Komponenten unter Verwendung der XML-basierten Sprache MyXaml durch Auslagerung der GUI-Definitionen zu realisieren und die Möglichkeit der Durchführung eines einfachen Brandings mit Hilfe eines simplen Werkzeuges zu demonstrieren. Die resultierende Anwendung sollte in jedem Fall auf der Plattform Pocket PC lauffähig sein.

Diese Anforderung der Funktionsfähigkeit des neugestalteten ESG-Client auf einem PDA stellte die eigentliche Problematik dar, da es zwar bereits verschiedene Lösungsansätze zur Trennung von Darstellungsebene und Applikationslogik gibt, z.B. Mozilla's XUL und Microsoft's XAML, diese jedoch nicht für die Plattform Pocket PC zur Verfügung stehen. Daher wurde im Rahmen der vorliegenden Diplomarbeit der Versuch unternommen, die Modifikation des bestehenden ESG-Client-Prototypen unter Verwendung der neuartigen MyXaml-Technologie umzusetzen. MyXaml stellt eine generisches Werkzeug zur Instantiierung beliebiger Klassen durch Beschreibung der zu erzeugenden Interaktionselemente in Form von Textauszeichnung dar und bietet neben der Möglichkeit der Anwendung auf einem Pocket PC zusätzlich die Möglichkeit der Verwendung beliebiger Steuerelemente sowie die Definition von Stylesheets und Skins.

Die erzielten Ergebnisse sollen im Folgenden kurz zusammengefasst werden:

- Der MyXaml-Parser wurde entsprechend der benötigten Funktionalität modifiziert. Dazu wurden fehlerhafte Implementierungen korrigiert sowie zusätzliche Funktionen hinzugefügt.
- Das in allen Zapping-Plug-Ins identisch implementierte Informationsfenster mit der zugehörigen MessageBox wurde als separate Komponente im Namensraum InfoPanel ausgelagert.
- Die Komponente ImageButton wurde modifiziert und somit die notwendige

Alle Rechte vorbehalten

Kompatibilität mit der MyXaml-Technologie erreicht, so dass ein ImageButton in der resultierenden Version des ESG-Client mittels Textauszeichnung beschrieben werden konnte.

- Es konnte eine weitgehende Skinningfähigkeit des ESG-Client selbst, des Startbildschirms, der Zapping-Plug-Ins für TV, Audio und Services sowie für das ausgelagerte InfoPanel inklusive der MessageBox realisiert werden. Die Angabe benutzerdefinierter Bilder innerhalb von Stildefinitionen konnte aufgrund der fehlenden Unterstützung durch den Parser nicht wie gewünscht realisiert werden. Eine einfache Austauschmöglichkeit der verwendeten Bilder besteht jedoch.
- Zur Umsetzung der Skinningfähigkeit wurden die jeweiligen GUI-Definitionen in separate MyXaml-Dokumente ausgelagert und der C#-Programmcode darauf abgestimmt.
- Allen verwendeten Interaktionselementen wurden individuelle Stile zugewiesen und diese in Form von MyXaml-Markup definiert, zu Skins zusammengefasst und in Stylesheet-Dokumenten abgelegt.
- Ein einfaches Skinning-Feature wurde implementiert und mit dessen Hilfe die erzielte Skinningfähigkeit der neugestalteten Anwendung getestet und Fehler in der Umsetzung aufgedeckt und behoben.

Ein Vergleich der zuvor genannten erzielten Ergebnisse mit den gestellten Anforderungen (siehe 4.6.) und der Zieldefinition (siehe 4.7.) zeigt, dass der modifizierte ESG-Client den Erwartungen gemäß umgesetzt wurde, auch wenn einige wenige Funktionen in der Kürze der Zeit nicht optimal implementiert werden konnten. Die MyXaml-Technologie hat sich als ein nützlicher und leicht zu verwendender[27] Lösungsansatz erwiesen, der jedoch ebenso weiterer Verbesserung bedarf, um die gewünschte Funktionalität vollständig in der neugestalteten Version des ESG-Client implementieren zu können. Besonders vorteilhaft waren die Möglichkeit der Beschreibung selbst erstellter Komponenten mittels Markup sowie die zur Verfügung gestellten Style- und MxComponent-Elemente, welche erst die Arbeit mit Stylesheets und Skins ermöglichten.

Da es sich bei der resultierenden Anwendung noch immer um eine prototypische Implementierung handelt, die nicht frei von Fehlern ist, bietet sich eine Weiterführung der vorliegenden Arbeit an. Im Hinblick auf eine Verbesserung der erzielten Ergebnisse müsste

27 Die unzureichende Dokumentation wird an dieser Stelle außer Acht gelassen. Die MyXaml-Technologie ist einfach zu verwenden, sofern erst einmal die grundlegenden Funktionen bekannt sind.

Alle Rechte vorbehalten

u.a. die zum Teil etwas unflexible Berechnung von Größen und Positionen der einzelnen Interaktionselemente dynamischer, auf den benutzerdefinierten Angaben im Skin basierend umgesetzt werden. Weiterhin sollte die Funktionalität der Style-Eigenschaft des MyXaml-Parsers dahingehend modifiziert werden, dass diese die Zuweisung von Bildern an die jeweiligen Interaktionselemente unterstützt. Für das Branding durch TSI-Mitarbeiter wäre ein Werkzeug zur Erstellung von Skins im *WYSIWYG*[28]-Modus für eine einfachere Anpassung der GUI vorteilhaft, da während der Veränderung der Stildefinitionen direkt die Auswirkungen auf die Darstellung der Elemente zu sehen sind. Dieses externe Werkzeug könnte auch die Zuweisung der erstellten Stylesheets und Skins übernehmen.

Das Hauptaugenmerk bei einer erneuten Modifikation des ESG-Client sollte jedoch auf der Verbesserung der Ablaufgeschwindigkeit während der alltäglichen Benutzung durch die Endkunden liegen, um wieder die Performanz der ursprünglichen Version zu erzielen. Dazu müsste zunächst eine Reihe von Tests durchgeführt werden, um herauszufinden, ob lediglich der Emulator zu langsam ist oder aber das Plug-In-Konzept bzw. eventuell sogar der XML-basierte Lösungsansatz verworfen werden muss.

Zusammenfassend lässt sich sagen, dass das Ziel der vorliegenden Arbeit erreicht und sämtliche Anforderungen, wenn auch teilweise nicht optimal, erfüllt werden konnten. Die erwartete Skinningfähigkeit wurde realisiert sowie durch einen Praxistest demonstriert. Das Branding durch einen TSI-Mitarbeiter mit grundlegenden Programmierkenntnissen (siehe 4.2.) sollte daher problemlos möglich sein. Die gewählte XML-basierte Lösung MyXaml eignete sich sehr gut zur Umsetzung der benutzerdefinierten Anpassbarkeit des ESG-Client. Die resultierende Anwendung ist weiterhin auf der Plattform Pocket PC lauffähig, allerdings aufgrund der Performanz kaum noch für die alltägliche Benutzung durch Endkunden geeignet.

Alle Rechte vorbehalten

28 Abkürzung für „What You See Is What You Get“

Anhang

A - Stilzuweisungen (zu 5.2.)

Zuweisungen individueller Stile an die einzelnen GUI-Elemente des ESG-Client:

Abbildung 35: Zuweisung von Stilen an die GUI-Elemente des Startscreens

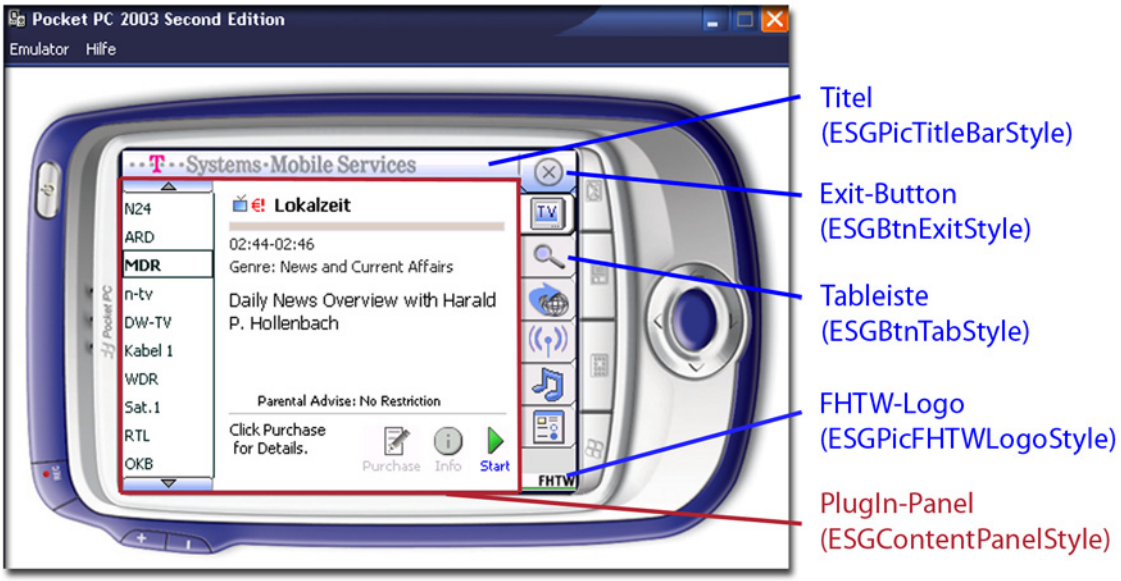

Abbildung 36: Zuweisung von Stilen an die GUI-Elemente des ClientMainForms

Abbildung 37: Zuweisung von Stilen an die GUI-Elemente des TVZapPlugIns und InfoPanels

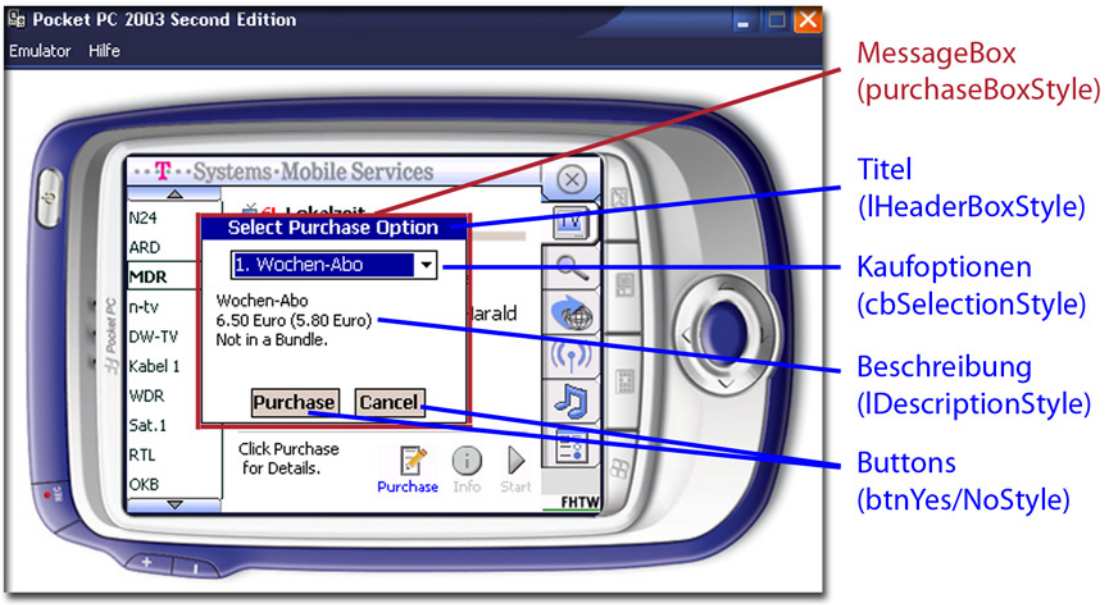

Abbildung 38: Zuweisung von Stilen an die GUI-Elemente der MessageBox des InfoPanels

B - Klassendiagramme des resultierenden ESG-Client (zu 6.1.3.)

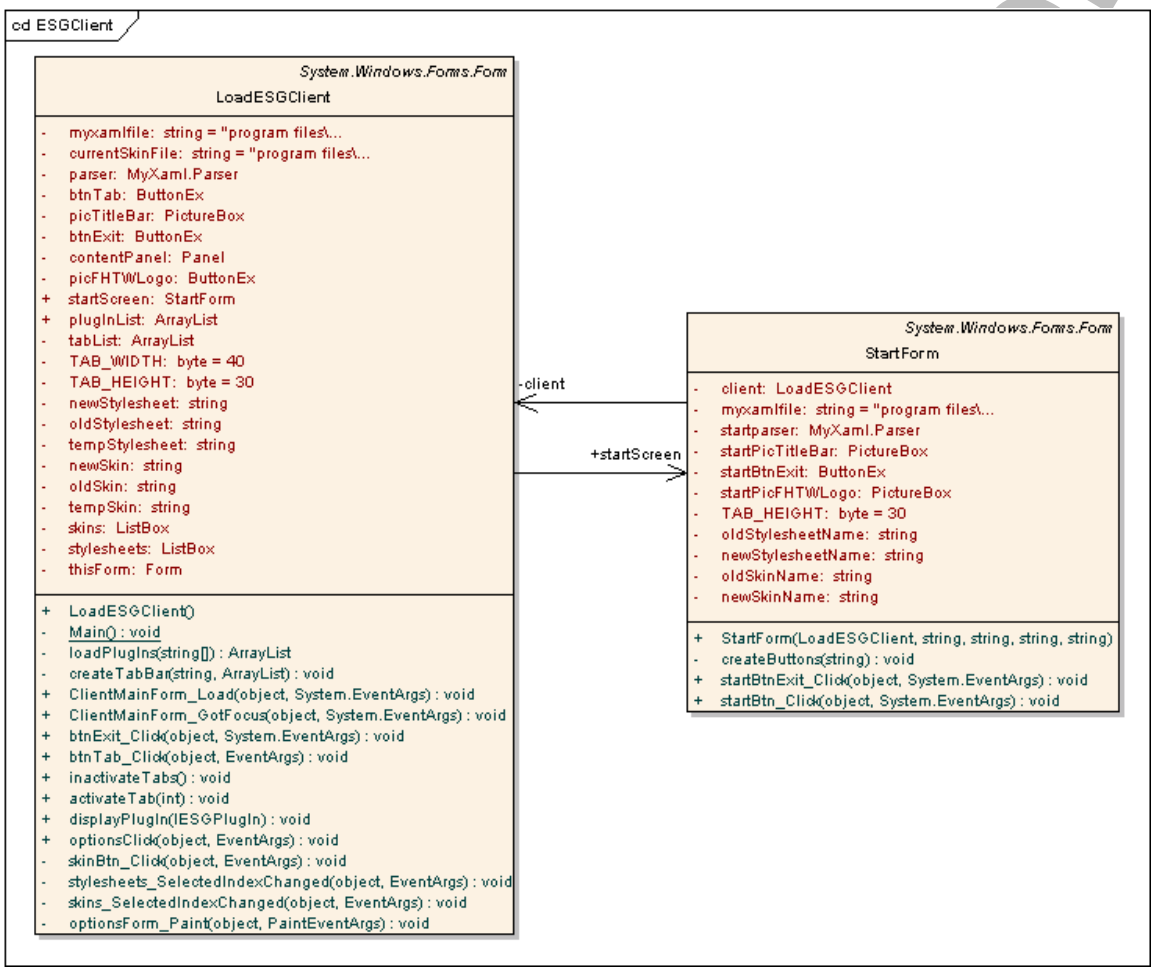

Abbildung 39: Klassendiagramm des modifizierten ESGClient-Packages

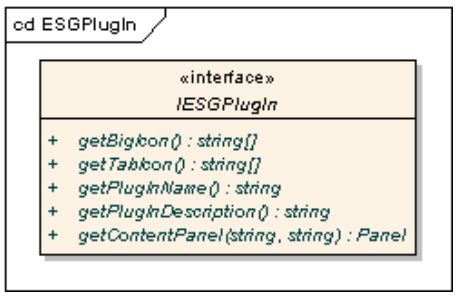

Abbildung 40: Klassendiagramm des modifizierten ESGPlugIn-Packages

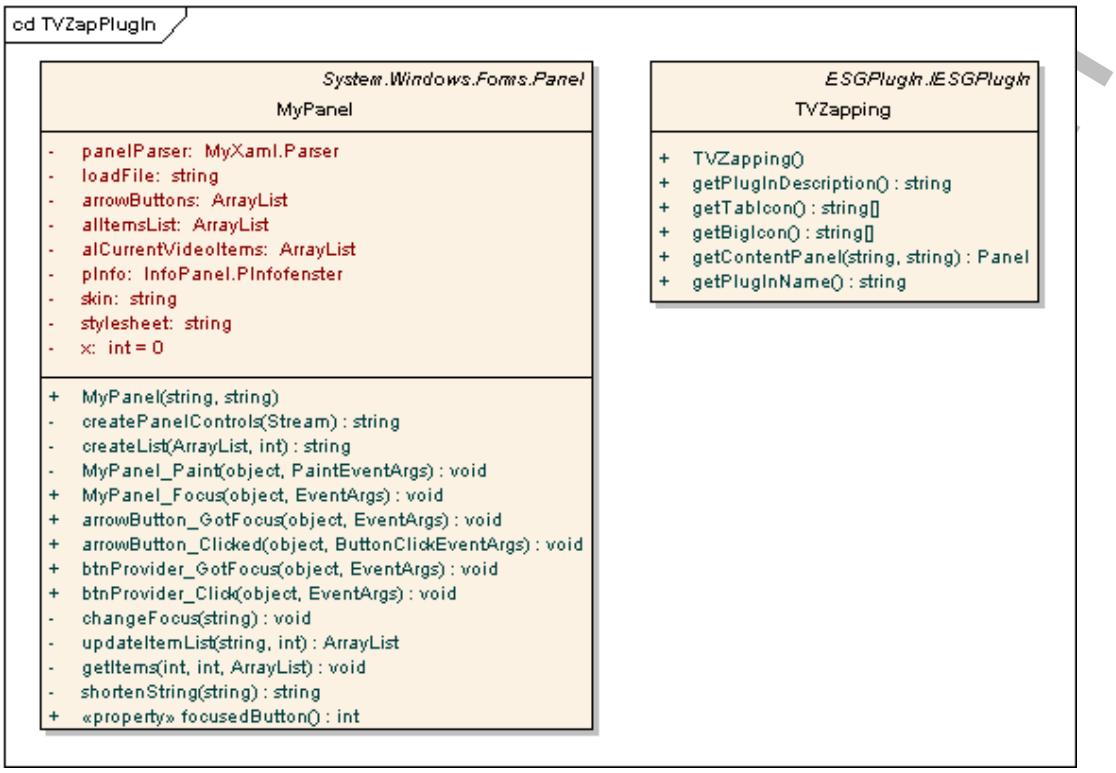

Abbildung 41: Klassendiagramm des modifizierten TVZapPlugIn-Packages

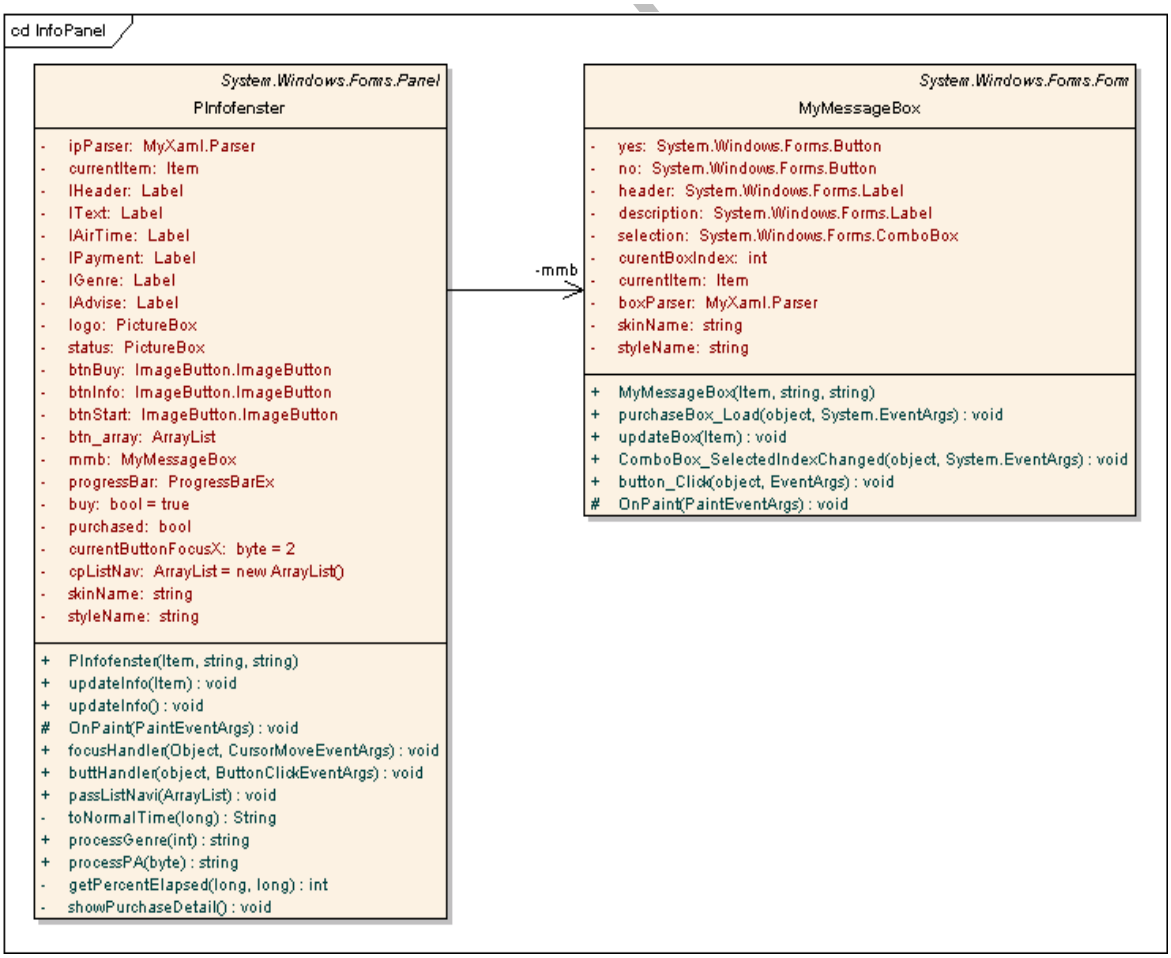

Abbildung 42: Klassendiagramm des InfoPanel-Packages

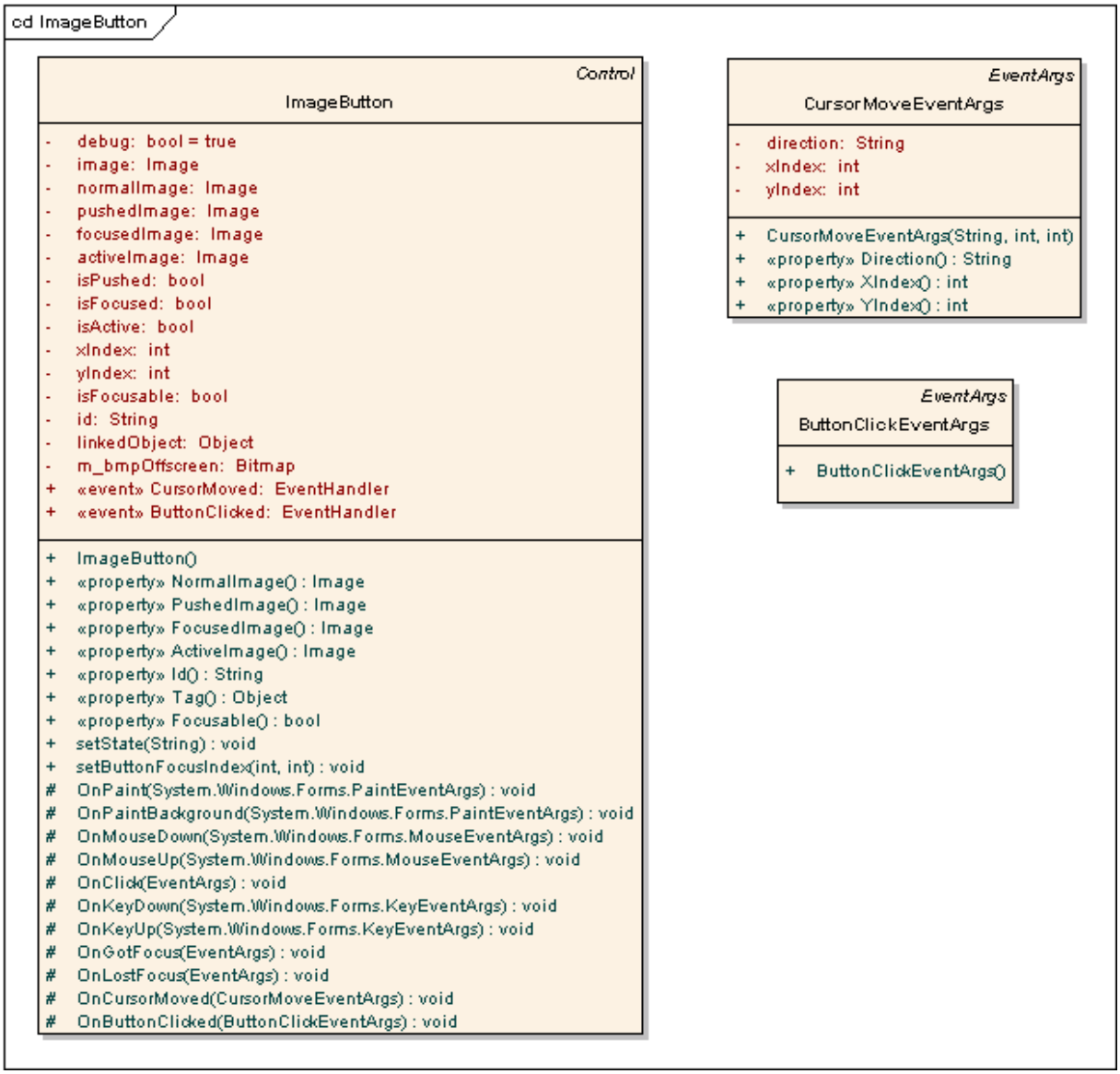

Abbildung 43: Klassendiagramm des modifizierten ImageButton-Packages

Alle Re

C - Abkürzungsverzeichnis

.NET	.NET Framework
.NET CF	.NET Compact Framework
CLR	Common Language Runtime
DOM	Document Object Model
DTD	Document Type Definition
DVB	Digital Video Broadcast
DVB-C	Digital Video Broadcast-Cable
DVB-H	Digital Video Broadcast-Handheld
DVB-H AL	Digital Video Broadcast-Handheld Abstraction Layer
DVB-S	Digital Video Broadcast-Satellite
DVB-SI	Digital Video Broadcast-Service Information
DVB-T	Digital Video Broadcast-Terrestrial
EPG	Electronic Program Guide
ESG	Electronic Service Guide
ETSI	European Telecommunications Standards Institute
FCL	Framework Class Library
GUI	Graphical User Interface
IP	Internet Protocol
ISO	International Organization for Standardization
MPEG	Moving Picture Experts Group
MPEG-2 TS	MPEG-2 Transport Stream
MVC	Model-View-Controller
PDA	Personal Digital Assistant
PPC	Pocket PC
SGML	Standard Generalized Markup Language
SVG	Scalable Vector Graphics
TSI	T-Systems International GmbH
UI	User Interface
URI	Uniform Resource Identifier
VGA	Video Graphics Array
W3C	World Wide Web Consortium
XAML	Extensible Application Markup Language
XHTML	Extensible Hypertext Markup Language
XML	Extensible Markup Language
XUL	XML User Interface Language

Alle Rechte vorbehalten

D - Glossar

.NET

„Satz von Softwaretechnologien des Softwareherstellers Microsoft, der als Gegenpol zu Sun Microsystems' Java eingeführt wurde, und neben einer virtuellen Laufzeitumgebung aus einem Rahmenwerk (Framework) von Klassenbibliotheken (API) und Diensten besteht, die als Basis für Eigenentwicklungen dienen. Einfach gesprochen handelt es sich also um eine umfangreiche Programmierumgebung, die eine neue Generation von Programmen und damit auch der Programmierung einläuten soll. Keine der verwendeten Technologien ist völlig neu, aber in der Gesamtheit ist .NET eine Innovation. [...] Die .NET-Plattform stellt mit der Common Language Infrastructure (CLI) eine Basis zur Ausführung von Programmen, die mit unterschiedlichen Programmiersprachen erstellt wurden, her. Dies wird durch die Verwendung einer (objektorientierten) virtuellen Maschine und die Framework Class Library (FCL) – einer gemeinsamen Klassenbibliothek – erreicht.“ [net]

Assembly

siehe Klassenbibliothek

Benutzungsschnittstelle

„Der Teil eines Programms, der den Datenaustausch mit dem Benutzer durchführt“ [ui]

Branding

hier: eine Form von Skinning, allerdings mit dem Unterschied, dass in diesem Fall nicht der Benutzer das Aussehen der Benutzungsschnittstelle verändert sondern eine Anwendung an verschiedene Firmen mit jeweils unterschiedlichem Aussehen verkauft wird. Dabei wird der Skin an das Corporate Design, u.a. Logos und Farben, der einzelnen Firmen angepasst

C#

„objektorientierte Programmiersprache, die vom Softwarehersteller Microsoft im Rahmen seiner .NET-Initiative eingeführt wurde“ [csharp2]

Code-Behind

Programmcode in einer Assembly, welcher den in einem XML-basierten Dokument referenzierten Eventkontext bereitstellt

Dokument

„Eine nicht ausführbare Datei mit in der Regel durch Menschen lesbarem Inhalt, z.B. HTML-Dokument, Word-Dokument. Besteht aus Inhalt, Struktur, Informationen zur visuellen Darstellung und semantischen Informationen“ [jen]

Dokumenttyp-Definition

„Deklaration in SGML- und XML-Dokumenten, die die Struktur eines solchen Dokuments festlegt“ [dtd]

DVB

„Bezeichnet in technischer Hinsicht die standardisierten Verfahren zur Übertragung von digitalen Inhalten (Fernsehen, Radio, Mehrkanalton, Raumklang, interaktive Dienste wie MHP, EPG und

Alle Rechte vorbehalten

Teletext und weitere Zusatzdienste) durch digitale Technik“ [dvb]

DVB-C

„Eine Variante von DVB, die für die Übertragung von digitalen Mehrwertdiensten über Kabelanschluss verwendet wird“ [dvbc]

DVB-H

Standard zur Übertragung digitaler Dienste auf mobile Empfangsgeräte über terrestrische Antennen

DVB-H AL

Abstraktionsschicht zur Kommunikation zwischen dem ESG-Client und dem unterliegenden Datenmodell von TSI, die Objektlisten und Hierarchien dieser Objekte aus den empfangenen XML-Daten erzeugt und diese auf Abruf zur Visualisierung an den ESG-Client übergibt

DVB-S

„Spezielle Variante zur Übertragung von DVB per Satellit“ [dvbs]

DVB-T

„terrestrische, das heißt erdgebundene, Variante von DVB, die in Europa und anderen Staaten als Standard für die Übertragung von digitalem Fernsehen per Antenne verwendet (werden) wird“ [dvbt]

Generic Markup

„Begriff für das Konzept der Auszeichnung von Inhalten nach semantischen anstatt nach visuellen Gesichtpunkten“ [jen]

Grafische Benutzungsschnittstelle

„Derjenige Teil einer Software, den der Benutzer unmittelbar (vor sich auf dem Monitor bzw. der Anzeige) sieht“ [gui]

Inline

Programmcode, der innerhalb eines XML-basierten Dokumentes vorliegt

Interaktionselement

Bestandteile einer grafischen Benutzungsschnittstelle, mittels derer ein Benutzer auf die Anwendung einwirken kann, z.B. Druckschalter

IP Datacast

Broadcast-Technologie zur kosteneffektiven und effizienten Verbreitung digitaler Inhalte an eine Vielzahl von Empfangsgeräten (vgl. [ipdata1])

Klassenbibliothek

„Spezielle Form einer Programmbibliothek, womit eine Sammlung selbständiger Programmkomponenten (in diesem Fall die Klassen) gemeint ist, die für die Wiederverwendung vorgesehen sind“ [dll]

Markup

„Bezeichnete ursprünglich die handschriftlichen Anmerkungen eines Layouters am Rand eines Dokuments, die später im Satz berücksichtigt wurden. Auf diese Weise wurden das Seitenformat bestimmt, Seiteninhalte angeordnet sowie typografische Festlegungen getroffen. Das Markup beschränkte sich damals fast ausschließlich auf visuelle Aspekte eines Textes. Mit dem Konzept des generic markup richtete sich der Fokus auf die semantische Auszeichnung von Inhalten, wie sie

Alle Rechte vorbehalten

gerade bei SGML- und XML-Dokumenttypen üblich ist. Markierungen machen dabei eine Aussage über die Art der markierten Stelle“ [jen].

Metapher

„Übertragung von Konzepten und Handlungsweisen aus einem Bereich der gegenständlichen Welt auf Interaktionen am Rechner“ ([hein], S. 163)

Metasprache

„Sprache, mit der eine andere Sprache erklärt oder definiert wird, mit der also die Regeln (die Grammatik) für diese Sprache festgelegt werden“ [jen]

MyXaml

Generisches Werkzeug zur Instantiierung von Klassen durch Markup zur Laufzeit (vgl. [myxaml8])

Parser

„Programm, das entscheidet, ob eine Eingabe zur Sprache einer bestimmten Grammatik gehört [...] Während des Parsens erfolgt eine syntaktische Überprüfung der Eingangsdaten. Bei der Überprüfung wird in der Regel aus den Daten ein Ableitungsbaum (in Anlehnung an das Englische gelegentlich auch als Parse-Baum bezeichnet) aufgebaut, um die Daten anschließend weiterverarbeiten zu können (z.B. semantische Analyse, Codegenerierung in einem Compiler oder Ausführung durch einen Interpreter).“ [pars]

Plug-In

„Softwareprogramm, das in ein anderes Softwareprodukt integriert wird“ [jen]
hier: Eine Klassenbibliothek, in der mindestens eine Klasse die Schnittstelle IESGPlugIn implementiert und die ein Panel sowie die zugehörige Funktionalität an den aufrufenden ESG-Client übergibt

Pocket PC

„Bezeichnet sowohl ein Computer-Betriebssystem als auch eine Reihe von persönlichen digitalen Assistenten (PDA's), die mit dem Betriebssystem Pocket PC betrieben werden“ [ppc].

Skin

Sammlung von Bildern, Farben und anderen Einstellungen, die die Darstellung der Steuerelemente und demzufolge das Aussehen der grafischen Benutzungsschnittstelle festlegen (vgl. [skin])

Steuerelement

siehe Interaktionselement

Stylesheet

hier: Dokument, das Skin-Definitionen enthält

Tag

„In Auszeichnungssprachen (engl. mark-up languages) wie SGML, XML oder HTML bezeichnet das Wort Tag die in Kleiner- und Größerzeichen eingeschlossenen Kürzel, die beispielsweise in HTML dazu dienen, Textelemente zu formatieren und damit deren Darstellung in Webseiten zu beeinflussen, oder aber, wie in XML, verwendet werden Daten zu klassifizieren und strukturieren. Dabei kommen so genannte öffnende und schließende Tags zum Einsatz. Letztere sind durch einen führenden Schrägstrich gekennzeichnet“ [tag].

Alle Rechte vorbehalten

E - Abbildungsverzeichnis

Abbildung 1: Aufbau des DVB-H-Systems ([frisch], S. 14).....7
Abbildung 2: OnAir-Darstellung des ESG-Client-Prototypen.....8
Abbildung 3: Zusammenspiel der einzelnen Komponenten des ESG-Client.....10
Abbildung 4: Aufbau des .NET Frameworks ([vasters], S. 8).....15
Abbildung 5: Dialoge mit Fenstersystemen.....17
Abbildung 6: Beispiel für ein einfaches XML-Dokument.....21
Abbildung 7: Beispiel für ein einfaches XAML-Dokument [xaml].....24
Abbildung 8: Beispiel für die Verwendung von Styles in XAML (vgl. [long4]).....24
Abbildung 9: Beispiel für ein MyXaml-Dokument.....26
Abbildung 10: Beispiel für eine MyXaml-Komponente.....27
Abbildung 11: Beispiel für den zu den Abbildungen 9 und 10 gehörigen C#-Code.....28
Abbildung 12: Systemarchitektur des bestehenden ESG-Client.....37
Abbildung 13: Screenshots des bestehenden StartForms (links) und ClientMainForms (rechts).....38
Abbildung 14: Klassendiagramm ESGClient-Package.....40
Abbildung 15: Klassendiagramm ESGPlugIn-Package.....41
Abbildung 16: Screenshot des bestehenden TVZapPlugIns.....42
Abbildung 17: Klassendiagramm TVZapPlugIn-Package.....43
Abbildung 18: Screenshot der MessageBox des PInfofensters.....45
Abbildung 19: Klassendiagramm ImageButton-Package.....46
Abbildung 20: Anwendungsfall-Diagramm “Branding“.....49
Abbildung 21: Entwurf zur Umsetzung einer GUI-Logik-Trennung.....57
Abbildung 22: Beispiel für ein MyXaml-Dokument zur Beschreibung der StartForm-GUI.....58
Abbildung 23: Beispiel für die MyXaml-Beschreibung eines ImageButtons.....59
Abbildung 24: Package-Diagramm des modifizierten ESG-Client (Entwurf).....61
Abbildung 25: Entwurf eines einfachen Skinning-Features.....62
Abbildung 26: Entwurf eines MyXaml-Stylesheet-Dokumentes mit einem Skin.....63
Abbildung 27: Geplante Funktionsweise des modifizierten ESG-Client (vgl. [myxaml7]).....66
Abbildung 28: Screenshot des implementierten Skinning-Features.....74
Abbildung 29: Beispiel für einen neuen Skin des Startbildschirms.....75
Abbildung 30: Package-Diagramm des resultierenden modifizierten ESG-Client.....76
Abbildung 31: Screenshot des Startbildschirms mit TestUserSkin (fehlerhaft).....82
Abbildung 32: ClientMainForm, TVZapPlugIn und InfoPanel mit TestUserSkin (fehlerhaft).....83
Abbildung 33: Screenshot des Startbildschirms mit TestUserSkin.....84
Abbildung 34: ClientMainForm, TVZapPlugIn und InfoPanel mit TestUserSkin.....85
Abbildung 35: Zuweisung von Stilen an die GUI-Elemente des Startscreens.....94
Abbildung 36: Zuweisung von Stilen an die GUI-Elemente des ClientMainForms.....94
Abbildung 37: Zuweisung von Stilen an die GUI-Elemente des TVZapPlugIns und InfoPanels.....95
Abbildung 38: Zuweisung von Stilen an die GUI-Elemente der MessageBox des InfoPanels.....95
Abbildung 39: Klassendiagramm des modifizierten ESGClient-Packages.....96

Alle Rechte vorbehalten

Abbildung 40: Klassendiagramm des modifizierten ESGPlugIn-Packages.....96
Abbildung 41: Klassendiagramm des modifizierten TVZapPlugIn-Packages.....97
Abbildung 42: Klassendiagramm des InfoPanel-Packages.....97
Abbildung 43: Klassendiagramm des modifizierten ImageButton-Packages.....98

Alle Rechte vorbehalten

F - Tabellenverzeichnis

Tabelle 1: Pocket PC-Übersicht (vgl. [wince1] und [wince2])....14
Tabelle 2: Anwendungsfall 1 – Hinzufügen eines neuen Stylesheet-Dokumentes....50
Tabelle 3: Anwendungsfall 2 – Hinzufügen eines neuen Skins....50
Tabelle 4: Anwendungsfall 3 – Modifizieren eines Skins....51
Tabelle 5: Anwendungsfall 4 – Zuweisen eines Stylesheet-Dokumentes....51
Tabelle 6: Anwendungsfall 5 – Zuweisen eines Skins....52

Alle Rechte vorbehalten

G - Literaturverzeichnis

[coombs]
Coombs, J.; Renear, A.; DeRose, S. (1987). MARKUP SYSTEMS AND THE FUTURE OF SCHOLARLY TEXT PROCESSING (PDF-Dokument, Stand: November 1987).
Internet: http://portal.acm.org/citation.cfm?id=32209&coll=ACM&dl=ACM&CFID=45074700&CFTOKEN=89599422 (Zugriff: 30.05.2005, 19:39 Uhr).

[csharp]
Drayton, P.; Albahari, B.; Neward, T.: C# In A Nutshell. 2nd Edition. Sebastopol: O'Reilly, 2003. – ISBN 0-596-00526-1

[csharp2]
Wikipedia-Autor (2005). C-Sharp - Wikipedia (WWW-Dokument, Stand: 27.06.2005).
Internet: http://de.wikipedia.org/wiki/C-Sharp (Zugriff: 28.06.2005, 00:13 Uhr).

[digi]
Lugmayr, A.; Niiranen, S.; Kalli, S.: Digital Interactive TV and Metadata - Future Broadcast Multimedia. New York: Springer-Verlag, 2004. – ISBN 0-387-20843-7

[dll]
Wikipedia-Autor (2005). Klassenbibliothek - Wikipedia (WWW-Dokument, Stand: 25.04.2005).
Internet: http://de.wikipedia.org/wiki/Klassenbibliothek (Zugriff: 28.06.2005, 12:26 Uhr).

[doc]
MyXaml-Wiki-Autoren (2005). MyXaml Wiki - Home (WWW-Dokument, Stand: 24.03.2005).
Internet: http://www.myxaml.com/wiki/ow.asp?Home (Zugriff: 07.06.2005, 00:25 Uhr).

[dom1]
Le Hégaret, P.; Whitmer, R.; Wood,L. (2005). W3C Document Object Model (WWW-Dokument, Stand: 19.01.2005). Internet: http://www.w3.org/DOM/ (Zugriff: 31.05.2005, 14:47 Uhr).

[dom2]
Le Hégaret, P.; Wood, L.; Robie, J. (2004). What is the Document Object Model? (WWW-Dokument, Stand: 07.04.2004).
Internet: http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/introduction.html (Zugriff: 31.05.2005, 15:04 Uhr).

[dtd]
Wikipedia-Autor (2005). Dokumenttypdefinition - Wikipedia (WWW-Dokument, Stand: 15.06.2005).
Internet: http://de.wikipedia.org/wiki/Dokumenttypdefinition (Zugriff: 22.06.2005, 16:35 Uhr).

[dvb]
Wikipedia-Autor (2005). Digital Video Broadcasting (WWW-Dokument, Stand: 30.04.2005).
Internet: http://de.wikipedia.org/wiki/Digital_Video_Broadcasting (Zugriff: 24.05.2005, 2:54 Uhr).

[dvbc]
Wikipedia-Autor (2005). DVB-C - Wikipedia (WWW-Dokument, Stand: 16.06.2005).
Internet: http://de.wikipedia.org/wiki/DVB-C (Zugriff: 22.06.2005, 16:28 Uhr).

[dvbs]
Wikipedia-Autor (2005). DVB-S - Wikipedia (WWW-Dokument, Stand: 18.06.2005).
Internet: http://de.wikipedia.org/wiki/DVB-S (Zugriff: 22.06.2005, 16:25 Uhr).

[dvbt]
Wikipedia-Autor (2005). DVB-T - Wikipedia (WWW-Dokument, Stand: 18.06.2005).
Internet: http://de.wikipedia.org/wiki/DVB-T (Zugriff: 22.06.2005, 16:22 Uhr).

Alle Rechte vorbehalten

[epg]
Wikipedia-Autor (2005). Elektronische Programmzeitschrift (WWW-Dokument, Stand: 05.05.2005).
Internet: http://de.wikipedia.org/wiki/Elektronische_Programmzeitschrift
(Zugriff: 24.05.2005, 11:37 Uhr).

[etsi1]
European Telecommunications Standards Institute (2004). ETSI EN 302 304 V1.1.1: Digital Video Broadcasting (DVB);Transmission System for Handheld Terminals (DVB-H)
(PDF-Dokument, Stand: November 2004).
Internet: http://webapp.etsi.org/exchangefolder/en_302304v010101p.pdf
(Zugriff: 24.05.2005, 10:57 Uhr).

[etsi2]
European Telecommunications Standards Institute (2003). ETSI EN 300 707 V1.2.1: Electronic Programme Guide (EPG);Protocol for a TV Guide using electronic data transmission (PDF-Dokument, Stand: April 2003). Internet: http://webapp.etsi.org/exchangefolder/en_300707v010201p.pdf
(Zugriff: 24.05.2005, 2:29 Uhr).

[etsi3]
European Telecommunications Standards Institute (2005). ETSI TR 102 377 V1.1.1: Digital Video Broadcasting (DVB); DVB-H Implementation Guidelines (PDF-Dokument, Stand: Februar 2005).
Internet: http://webapp.etsi.org/exchangefolder/tr_102377v010101p.pdf
(Zugriff: 28.06.2005, 14:02 Uhr).

[fcl]
Microsoft Corporation (2005). Klassenbibliothek (WWW-Dokument, Stand: 2005).
Internet: http://msdn.microsoft.com/library/deu/default.asp?url=/library/DEU/cpref/html/cpref_start.asp
(Zugriff: 28.06.2005, 12:22 Uhr).

[frisch]
Frischholz, R. (2004). Chancen und Grenzen von DVB-H (PDF-Dokument, Stand: 24.06.2004).
Internet: http://www.berlin.de/senwiarbfrau/projektzukunft/dvbt/ws_dvbh_04/t-systems.pdf
(Zugriff: 24.05.2005, 10:21 Uhr).

[gui]
Wikipedia-Autor (2005). Grafische Benutzeroberfläche - Wikipedia
(WWW-Dokument, Stand: 20.06.2005).
Internet: http://de.wikipedia.org/wiki/Grafische_Benutzeroberfl%C3%A4che
(Zugriff: 22.06.2005, 16:51 Uhr).

[hand]
Gievers, R.: Das Praxisbuch Pocket PC 2003 Second Edition. Ort unbekannt: Handit.de, 2004.
– ISBN 393803605-2

[hein]
Heinecke, A.: Mensch-Computer-Interaktion. München, Wien: Fachbuchverlag Leipzig im Carl Hanser Verlag, 2004. – ISBN 3-446-22591-9

[ipdata1]
IPDC Forum (2003). IPDC Forum - About IP Datacasting - FAQ (WWW-Dokument, Stand: 2003).
Internet: http://www.ipdc-forum.org/about/faq.html (Zugriff: 24.04.2005, 11:39 Uhr).

[ipdata2]
Broadcastpapers Pty Ltd (2004). BROADCASTPAPERS.COM: IP DATACAST TERMINALS: DESCRIPTION OF IMPLEMENTATION PRINCIPLES (WWW-Dokument, Stand: 2004).
Internet: http://www.broadcastpapers.com/data/NokiaIPDatacast03.htm
(Zugriff: 24.05.2005, 11:43 Uhr).

[jen]
Jendryschik, M. (2005). Webentwicklung: Glossar (WWW-Dokument, Stand: 13.03.2005).

Alle Rechte vorbehalten

Internet: http://jendryschik.de/wsdev/glossar/gesamt (Zugriff: 30.05.2005, 18:46 Uhr).

[kauf]
Kaufmann, B. (2001). Das Model-View-Controller Modell (PDF-Dokument, Stand: 30.10.2001).
Internet: http://fara.cs.uni-potsdam.de/~kaufmann/tuts/mvc.pdf (Zugriff: 24.05.2005, 12:21 Uhr).

[long1]
Chand, M. (2003). Longhorn for Developers - A Brief Introduction
(WWW-Dokument, Stand: 11.12.2003).
Internet: http://www.c-sharpcorner.com/Longhorn/Longhorn/LonghornIntroduction.asp
(Zugriff: 03.06.2005, 14:34 Uhr).

[long2]
Esposito, D. (2004). Code Name Longhorn: A First Look at Writing and Deploying Apps in the Next Generation of Windows (WWW-Dokument, Stand: Januar 2004).
Internet: http://msdn.microsoft.com/longhorn/default.aspx?pull=/msdnmag/issues/04/01/DevelopingAppsforLonghorn/default.aspx (Zugriff: 03.06.2005, 14:38 Uhr).

[long3]
Rector, B. (2003). The "Longhorn" Application Model (WWW-Dokument, Stand: Oktober 2003).
Internet: http://msdn.microsoft.com/Longhorn/understanding/books/rector/default.aspx?pull=/library/en-us/dnintlong/html/longhornch01.asp (Zugriff: 07.06.2005, 11:57 Uhr).

[long4]
Rector, B. (2003). Controls and XAML (WWW-Dokument, Stand: Dezember 2003).
Internet: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnintlong/html/longhornch03.asp (Zugriff: 07.06.2005, 12:53 Uhr).

[myxaml1]
Clifton, M. (2004). What Is XAML? (WWW-Dokument, Stand: 05.11.2004).
Internet: http://myxaml.com/marcclifton/archive/2004/11/05/524.aspx (Zugriff: 07.06.2005, 00:29 Uhr).

[myxaml2]
Clifton, M. (2004). MyXaml Version 1.00 Is Released! (WWW-Dokument, Stand: 31.08.2004).
Internet: http://myxaml.com/marcclifton/archive/2004/08/31/457.aspx (Zugriff: 07.06.2005, 00:31 Uhr).

[myxaml3]
Clifton, M. u.a. (2004). MyXAML (WWW-Dokument, Stand: 09.07.2004).
Internet: http://www.longhornblogs.com/akinney/archive/2004/03/17/2773.aspx
(Zugriff: 07.06.2005, 00:33 Uhr).

[myxaml4]
unbekannt (2005). myxaml.tigris.org (WWW-Dokument, Stand: 2005).
Internet: http://myxaml.tigris.org/ (Zugriff: 07.06.2005, 00:34 Uhr).

[myxaml5]
MyXaml (2005). MyXaml - Support (WWW-Dokument, Stand: 2005).
Internet: http://www.myxaml.com/support/faq.aspx (Zugriff: 07.06.2005, 00:35 Uhr).

[myxaml6]
Bierman, B. (2004). Using MyXaml on the Compact Framework
(WWW-Dokument, Stand: 02.09.2004).
Internet: http://www.codeproject.com/netcf/UsingMyXamlCF.asp (Zugriff: 07.06.2005, 00:38 Uhr).

[myxaml7]
Clifton, M. (2004). MyXAML--XAML-style gui generator (WWW-Dokument, Stand: 13.02.2004).
Internet: http://www.codeproject.com/cs/miscctrl/xmlGuiGenerator.asp
(Zugriff: 24.06.2005, 14:38 Uhr).

Alle Rechte vorbehalten

[myxaml8]
MyXaml-Wiki-Autor (2004). MyXaml Wiki - Introduction (WWW-Dokument, Stand: 28.06.2004).
Internet: http://www.myxaml.com/wiki/ow.asp?Introduction (Zugriff: 22.06.2005, 17:05 Uhr).

[net]
Wikipedia-Autor (2005). .NET - Wikipedia (WWW-Dokument, Stand: 25.06.2005).
Internet: http://de.wikipedia.org/wiki/.NET (Zugriff: 28.06.2005, 00:20 Uhr).

[netcf]
Microsoft Corporation (2005). .NET Compact Framework (WWW-Dokument, Stand: 2005).
Internet: http://msdn.microsoft.com/library/deu/default.asp?url=/library/DEU/dv_evtuv/html/etconnet-compactframework.asp (Zugriff: 24.05.2005, 12:24 Uhr).

[pars]
Wikipedia-Autor (2005). Parser - Wikipedia (WWW-Dokument, Stand: 08.06.2005).
Internet: http://de.wikipedia.org/wiki/Parser (Zugriff: 28.06.2005, 00:10 Uhr).

[ppc]
Wikipedia-Autor (2005). Microsoft Pocket PC - Wikipedia (WWW-Dokument, Stand: 14.06.2005).
Internet: http://de.wikipedia.org/wiki/Microsoft_Pocket_PC (Zugriff: 22.06.2005, 17:18 Uhr).

[refl]
Microsoft Corporation (2005). Ermitteln von Typinformationen zur Laufzeit
(WWW-Dokument, Stand: 2005).
Internet: http://msdn.microsoft.com/library/deu/default.asp?url=/library/DEU/cpguide/html/cpcondiscoveringtypeinformationatruntime.asp (Zugriff: 06.06.2005, 22:06 Uhr).

[richter]
Richter, J.: Microsoft .NET Framework Programmierung. Unterschleißheim: Mircosoft Press Deutschland, 2002. – ISBN 3-86063-650-2

[skin]
Wikipedia-Autor (2005). Skin (Computer) - Wikipedia (WWW-Dokument, Stand: 14.06.2005).
Internet: http://de.wikipedia.org/wiki/Skin_(Computer) (Zugriff: 24.06.2005, 13:31 Uhr).

[sysxml]
Microsoft Corporation (2005). System.Xml-Namespace (WWW-Dokument, Stand: 2005).
Internet: http://msdn.microsoft.com/library/deu/default.asp?url=/library/DEU/cpref/html/frlrfsystemxml.asp (Zugriff: 06.06.2005, 22:13 Uhr).

[tag]
Wikipedia-Autor (2005). Tag (Informatik) - Wikipedia (WWW-Dokument, Stand: 17.05.2005).
Internet: http://de.wikipedia.org/wiki/Tag_%28Informatik%29 (Zugriff: 22.06.2005, 17:12 Uhr).

[ui]
Wikipedia-Autor (2005). Benutzerschnittstelle - Wikipedia (WWW-Dokument, Stand: 15.06.2005).
Internet: http://de.wikipedia.org/wiki/Benutzerschnittstelle (Zugriff: 22.06.2005, 16:53 Uhr).

[vasters]
Vasters u.a.: .NET-Crashkurs. Unterschleißheim: Microsoft Press Deutschland, 2002.
– ISBN 3-86063-519-0

[west]
Westphal, R.: .NET kompakt. Heidelberg, Berlin: Spektrum Akademischer Verlag, 2002.
– ISBN 3-8274-1185-8

[wince1]
Wikipedia-Autor (2005). Microsoft Pocket PC (WWW-Dokument, Stand: 12.05.2005).
Internet: http://de.wikipedia.org/wiki/Windows_CE (Zugriff: 24.05.2005, 12:02).

Alle Rechte vorbehalten

[wince2]
NCP engineering GmbH (unbekannt). Windows CE Kompatibilität
(WWW-Dokument, Stand: unbekannt).
Internet: http://www.ncp.de/deutsch/services/cekompat/infoWCE300.html (Zugriff: 24.05.2005, 12:06).

[xaml]
Clifton, M. (2004). What Is XAML (WWW-Dokument, Stand: 2004).
Internet: http://myxaml.com/whitePapers/whatIsXAML.htm (Zugriff: 01.06.2005, 11:16 Uhr).

[xml]
Eckstein, R.: XML kurz & gut. Köln: O'Reilly Verlag, 2000. – ISBN 3-89721-219-6

Alle Rechte vorbehalten

H - CD-ROM

Auf der beigefügten CD-ROM befinden sich die folgenden Inhalte:

- Diplomarbeit als PDF-Dokument
- Programmcode des modifizierten ESG-Client
- Technische Dokumentation des modifizierten ESG-Client
- Benutzerhandbuch (inklusive Installationsbeschreibung)
- Anleitung zur Erstellung benutzerdefinierter Stylesheets und Skins
- Programmcode des ursprünglichen ESG-Client (inklusive der verwendeten Version des OpenNETCF SmartDeviceFrameworks)

Die CD-ROM verfügt über eine Autostart-Funktion und ermöglicht somit einen problemlosen Zugriff auf die einzelnen Inhalte über einen Menübildschirm.

Alle Rechte vorbehalten

I - Danksagung

Für die Betreuung während der Bearbeitung der vorliegenden Diplomarbeit bedanke ich mich recht herzlich bei Herrn Prof. Thomas Bremer.

Weiterhin gilt mein Dank den ehemaligen Teammitgliedern des DVB-H-Projektes, die mir im Rahmen dieser Arbeit bereitwillig zahlreiche Fragen zur Programmierung der zugrunde liegenden Anwendung beantwortet haben.

Meinen Dank möchte ich ebenso der freiwilligen Testperson sowie den Korrekturlesern dieser Arbeit aussprechen.

Für die seelische und moralische Unterstützung während der Erstellungsphase dieser Arbeit bedanke ich mich vor allem bei meinem Freund und meiner Familie sowie einigen guten Freunden.

Alle Rechte vorbehalten

J - Eigenständigkeitserklärung

Hiermit erkläre ich, Merrit Kröger, an Eides statt, dass ich die vorliegende Diplomarbeit mit dem Thema „Abstraktion von Benutzungsschnittstellen auf Basis der Extensible Markup Language (XML)“ selbstständig und ohne fremde Hilfe verfasst habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel verwendet und die diesen Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Berlin, 04. Juli 2005

Merrit Kröger

Alle Rechte vorbehalten